WOA 2006

Dagli Oggetti Agli Agenti

*Sistemi Grid, P2P e Self-**

Catania (Italia), 26-27 Settembre 2006

Atti delle giornate di lavoro a cura di:

Flavio De Paoli, Antonella Di Stefano
Andrea Omicini, Corrado Santoro

Organizzato da:
Associazione Italiana per l'Intelligenza Artificiale
Associazione Italiana Tecnologie Avanzate basate su concetti Orientati ad Oggetti

Con il patrocinio della Facoltà di Ingegneria dell'Università degli Studi di Catania

PREFAZIONE

Le tecnologie degli agenti stanno assumendo un ruolo centrale non solo nel settore dell'intelligenza artificiale, ma anche in settori più tradizionali dell'informatica quali l'ingegneria del software e i linguaggi di programmazione, dove il concetto di agente viene considerato una naturale estensione di quello di oggetto. L'importanza di queste tecniche è dimostrata anche in campo industriale dall'interesse per il loro utilizzo nella realizzazione di strumenti e applicazioni in molteplici aree.

Ormai giunto alla sua settima edizione, il workshop "WOA: dagli Oggetti agli Agenti" costituisce un'usuale occasione di scambio di idee per tutti i ricercatori che operano nell'ambito dei sistemi ad agenti. L'edizione 2006, svoltasi dal 26 al 27 Settembre presso la Facoltà di Ingegneria dell'Università di Catania, è stata patrocinata dall'Associazione Italiana per l'Intelligenza Artificiale (AI*IA) e dall'Associazione Italiana Tecnologie Avanzate Basate su concetti Orientati ad Oggetti (TABOO). Questa edizione ha avuto come tema principale i Sistemi GRID, Self-* e Peer-to-Peer, anche se gli articoli presentati hanno poi spaziato su varie tematiche relative al mondo degli agenti software. Una novità introdotta quest'anno è stata la "sessione demo", uno spazio riservato a tutti coloro che desideravano dimostrare ai partecipanti al workshop i propri prototipi di sistemi software ad agenti. Inoltre il workshop è stato preceduto, il giorno 25 Settembre 2006, da una miniscuola per studenti di dottorato e laureandi che ha consentito alle nuove leve della ricerca di conoscere nel dettaglio e discutere produttivamente alcune tra le tematiche principali del mondo della ricerca sui sistemi ad agenti.

Il Comitato Scientifico Organizzatore desidera esprimere un vivo ringraziamento a tutti coloro che hanno contribuito al successo di questa settima edizione di WOA: gli autori degli articoli inviati, i componenti del comitato di programma per il lavoro di revisione, la Facoltà di Ingegneria dell'Università di Catania, gli organizzatori locali e tutti collaboratori che hanno partecipato all'organizzazione.

IL COMITATO SCIENTIFICO ORGANIZZATORE
Flavio De Paoli, Antonella Di Stefano
Andrea Omicini, Corrado Santoro

Comitato Scientifico Organizzatore

Flavio De Paoli	Università di Milano - Bicocca
Antonella Di Stefano	Università di Catania
Andrea Omicini	Università di Bologna - Cesena
Corrado Santoro	Università di Catania

Organizzazione Sessione Demo

Alfredo Garro	Università della Calabria

Coordinatore Miniscuola

Matteo Baldoni	Università di Torino

Comitato Organizzzatore Locale

Antonella Di Stefano	Università di Catania
Giuseppe Pappalardo	Università di Catania
Corrado Santoro	Università di Catania
Emiliano Tramontana	Università di Catania

Comitato di Programma

Stefania Bandini	Università di Milano - Bicocca
Pietro Baroni	Università di Brescia
Carlo Bellettini	Università di Milano
Federico Bergenti	Università di Parma
Lorenzo Bettini	Università di Firenze
Enrico Blanzieri	Università di Trento
Paolo Bouquet	Università di Trento e IRST
Nadia Busi	Università di Bologna
Giacomo Cabri	Università di Modena e Reggio Emilia
Marco Cadoli	Università di Roma La Sapienza
Nicola Cannata	Università di Camerino
Massimo Cossentino	ICAR-CNR, Palermo
Francesco Donini	Università di Tuscia - Viterbo
Rino Falcone	ISTC-CNR, Roma
Giancarlo Fortino	Università di della Calabria
Alfredo Garro	Università di della Calabria
Laura Giordano	Università di Piemonte Orientale
Letizia Leonardi	Università di Modena e Reggio Emilia
Marco Mamei	Università di Modena e Reggio Emilia
Sara Manzoni	Università di Milano - Bicocca
Viviana Mascardi	Università di Genova
Emanuela Merelli	Università di Camerino
Rebecca Montanari	Università di Bologna
Maria Teresa Pazienza	Università di Roma - Tor Vergata
Paola Quaglia	Università di Trento
Alessandro Ricci	Università di Bologna - Cesena
Giovanni Rimassa	Whitestein Technologies, Zurich, Switzerland
Carla Simone	Università di Milano - Bicocca
Paolo Torroni	Università di Bologna
Emilio Tuosto	University of Leicester, UK
Eloisa Vargiu	Università di Cagliari
Mario Verdicchio	Politecnico di Milano
Mirko Viroli	Università di Bologna - Cesena
Giuseppe Vizzari	Università di Milano - Bicocca

INDICE

simpA-WS: An Agent-Oriented Computing Technology forWS-based SOA Applications

Alessandro Ricci, Claudio Buda, Nicola Zaghini, Antonio Natali, Mirko Viroli, Andrea Omicini
DEIS, Università di Bologna Alma Mater Studiorum
Cesena (FC), 47023 Italy
Emails: {a.ricci, claudio.buda, nicola.zaghini, antonio.natali, mirko.viroli, andrea.omicini}@unibo.it

***Abstract*— This document briefly describes simpA-WS, a Java-based agent-oriented computing technology to flexibly and effectively implement WS-I compliant SOA/WS applications—i.e. Web-Service applications with a Service-Oriented Architecture—both on the user side and the service side.**

I. Introduction

simpA-WS is a Java-based technology that makes it possible to build WS-I SOA/WS compliant applications adopting an agent-oriented style in designing and developing the systems. simpA-WS is based on of simpA model and technology [1], an *agent-oriented* extension of the Java Object-Oriented computational model providing high-level abstractions for designing and developing complex software systems.

On the one side, simpA-WS provides a framework API to build *user applications* in terms of sets of agents that flexibly interact and use Web Services compliant with the WS-I Basic Profile [12], represented as *artifacts* in agent workspaces. On the other side, simpA-WS provides an API framework and a middleware for building WS-I compliant Web Services in terms of set of agents as providers of the services.

A. Human Cooperative Working Environments as Background Metaphor

To tackle the complexity of modern and future software systems, simpA introduces *agents* and *artifacts* as high-level abstractions to design and build distributed / concurrent software systems. This is the A&A (Agents and Artifact) conceptual model [16], [15], [10], based on inter-disciplinary studies involving Activity Theory and Distributed Cognition as main conceptual background frameworks [9].

A&A metaphors are taken from human cooperative working environments, where "systems" are composed by individual autonomous entities which pro-actively carry on some kind of activities or tasks, both individual and cooperative, typically requiring forms of interaction and coordination with other individuals. A fundamental aspect of such a picture is the context —or the *environment*— that makes it possibile to such activities to take place. *Artifacts* designed and built by humans are an essential part of such a context, both as outcome of the activities and as the *tools* exploited by humans to support and realise them. Artifacts can be then resources and objects constructed during the activities, but also whatever tools is used to support humans communication, coordination, and—more generally—cooperative working activities. The overall picture is given by *workspaces* where ensembles of individuals work and interact in a coordinated manner, by communicating and sharing / using the same artifacts, so as to achieve some kind of objective.

A&A brings this metaphor down to software engineering, conceiving a software system as one or multiple workspaces where ensembles of autonomous entities—the agents—execute their working activities and interact by co-constructing, sharing and using artifacts, analogously to the human case.

simpA provides agents and artifacts as basic high-level building blocks to decompose and structure complex systems, in particular:

- agents are provided as first-class abstractions to model and design *pro-active* entities, i.e. entites programmed so as to autonomously execute some kind activity—composed by one or more *tasks*—encapsulating the control of such activities;
- artifacts are provided as first-class abstractions to model and design what is used or constructed by agents during their activities, including resources, tools, devices: any *passive* entity encapsulating some kind of functionality, exposed by a proper interface;
- *workspaces* are the logical place where agents and artifacts are immersed, used to give a topology to the overall activities, and then partition the application environment.

Objects and classes are used as basic abstractions to define data structures to build agents and artifacts.

B. A&A *for Implementing SOA / Web Service Applications*

simpA-WS makes it possible to exploit the A&A approach and simpA for implementing SOA and Web-Service applications, following the basic architectured described in W3C documentation [17](see Fig. 1).

Using simpA-WS both service users and providers are modelled as simpA agents, as pro-active entities that respectively *(i)* need to access and use services in their working activities, encapsulating the business logic of user applications, and *(ii)* process the requests and messages for services, encapsulating the service business logic. In both cases, artifacts are used as high-level mediating entities functioning as interfaces, encapsulating the technology needed to enable the interaction using WS-* standards. In particular (refer to Fig. 2 and Fig. 3):

- on the user side, a specific kind of artifacts – called WS-Artifact — is provided to make it possible for simpA

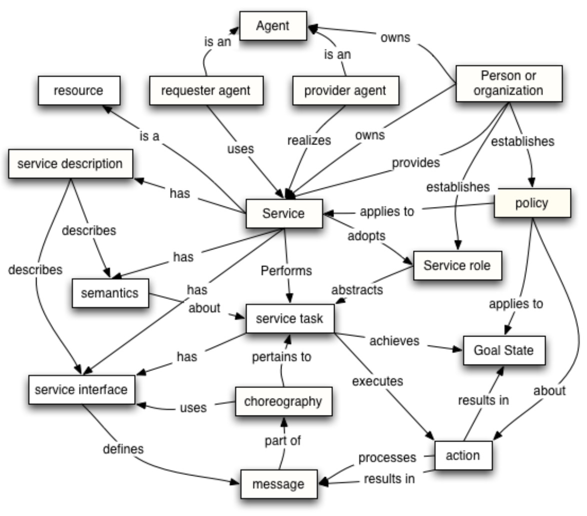


Fig. 1. Web Service Service Model according to W3C

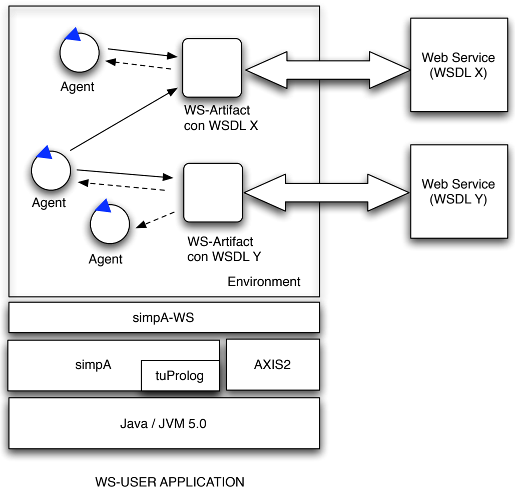


Fig. 2. Abstract architecture of simpA-WS user applications

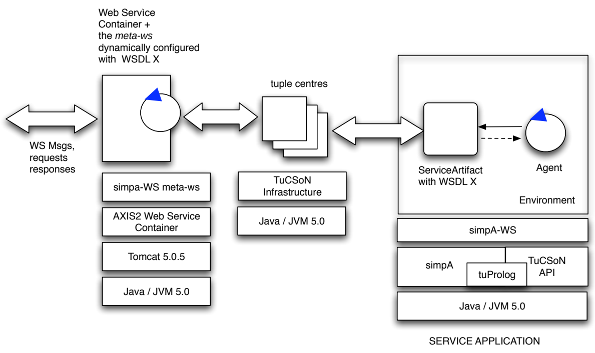


Fig. 3. Abstract architecture of simpA-WS service applications

agents to access and use what ever existing Web Service, by simply creating and using instances of such an artifact;

- on the service side, artifacts called Service-Artifacts are provided to make it possible for simpA agents to get and process the messages delivered to a specific Web Service, again by simply creating and using instances of such an artifact.

The adoption of the agent level of abstraction, and in particular of agents and artifacts basic building blocks, makes it possible to exploit a fully uncoupled approach for modelling and realising interaction with Web Services, as required by true service-oriented architecture [6]. On the one side, agents *use* Web Services by executing operations on WS-Artifact artifacts, by means of fully asynchronous *actions*. On the other side, agents *perceive* possible information or result generated Web Services by observing events — through simpA *sensing* primitives — generated by such artifacts.

II. System Requirements

Currently simpA-WS can run on any Java-based platform, version 5.0. Consequently, it can be seamlessly deployed on machines with different kind of operating systems and hardware configuration. A light-weight version is planned, in order to port simpA-WS—the API for implementing user application in particular—on J2ME, the Java platform for mobile devices, so as to implement user applications on top of J2ME-enabled smart phones / devices.

III. Design and Implementation

Fig. 2 and Fig. 3 provide an overview of the architecture of the simpA-WS technology, respectively on the user side and the service side. Fig. 4 shows a mixed and articulated case, where services are themselves users of other services.

simpA and simpA-WS are fully developed in Java, and exploit apache Axis2 open-source libraries [7] for realising low-level SOAP-based interaction with Web Services on the user side and as Web Service container on the service side, on top of Tomcat apache Java-based Web Server [8].

simpA and simpA-WS internally exploits tuProlog [5] and TuCSoN [11] research technologies, respectively a light-weight Java-based Prolog interpreter and a tuple-based agent coordination infrastructure.

IV. A Sample Application

Among the examples provided with simpA-WS distribution, a supply-chain sample application is provided, following the reference sample application defined by WS-I organisation, available among the deliverables at WS-I Web Site [13]. The supply-chain example is one among the illustrative scenarios defined by the WS-I Sample Applications Working Group to show the benefits of having interoperable Web services applications, and to demonstrate the application of the WS-I profiles to those scenarios.

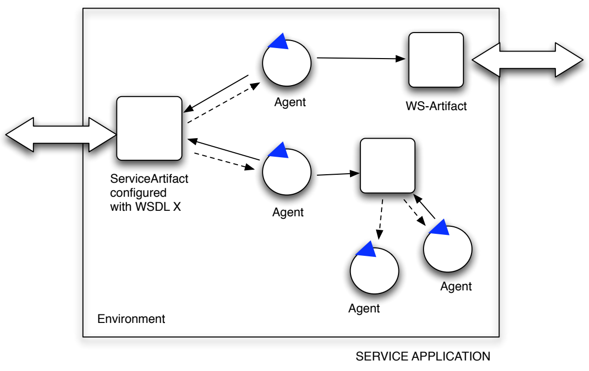


Fig. 4. Abstract architecture of mixed simpA-WS applications

V. simpA-WS in Real-World and Industrial Applications

simpA-WS is one of the technologies experimented for implementing SOA-based applications in the context of STIL ("Strumenti Telematici per l'Interoperabilità delle reti di imprese: Logistica digitale integrata per l'Emilia-Romagna").

STIL is a 2-years project funded by Emilia-Romagna, in the context of the "Iniziativa 1.1 del Piano Telematico Regionale" initiative [14]. The project has been funded to push and improve the research activities in Emilia-Romagna targeted to exploit innovative ICT technologies for the creation of a global digital logistic district. Among the objectives, STIL is dedicated to the creation of virtual organizations grouping together different kind of actors directly or indirectly involved in the logistic supply-chain, providing them effective ICT supports for integrating and innovating their business.

For the purpose, a SOA-based infrastructure has been conceived, designed and implemented to enable interoperability among the different participants. The STIL infrastructure is meant to provide an effective support for enabling communication, coordination and cooperation among the open and heterogeneous kind of WS-based applications and services. simpA-WS is currently experimented as one of the state-of-the-art technologies for implementing the applications and services, and first results are available on STIL web sites [14], [3].

VI. Industrial Support and Distribution

simpA and simpA-WS are open-source projects, and can be freely downloaded and used for research and non-commercial purposes from related web sites [1], [2]. Besides the open-source prototypes, an industrial-version of the technology will be available as commercial product distributed by IRIS [4], a start-up spin-off company hosted in Cesena.

References

[1] The aliCE Research Group. simpA official web site. http://www.alice.unibo.it/projects/simpa.
[2] The aliCE Research Group. simpA-WS official web site. http://www.alice.unibo.it/projects/simpa-ws.
[3] aliCE-Unibo research unit. The STIL-UNIBO project web site. http://www.alice.unibo.it/projects/stil.
[4] The IRIS Company. IRIS official web site. http://www.irislab.eu.
[5] Enrico Denti, Andrea Omicini, and Alessandro Ricci. Multi-paradigm Java-Prolog integration in tuProlog. *Science of Computer Programming*, 57(2):217–250, August 2005.
[6] Thomas Erl. *Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004.
[7] The Apache Software Foundation. Axis 2.0 — next generation web services — official web site. http://ws.apache.org/axis2/.
[8] The Apache Software Foundation. Tomcat official web site. http://tomcat.apache.org/.
[9] B. A. Nardi. *Context and Consciousness: Activity Theory and Human-Computer Interaction*. MIT Press, 1996.
[10] Andrea Omicini, Alessandro Ricci, and Mirko Viroli. *Agens Faber*: Toward a theory of artefacts for MAS. *Electronic Notes in Theoretical Computer Sciences*, 150(3):21–36, 29 May 2006. 1st International Workshop "Coordination and Organization" (CoOrg 2005), COORDINATION 2005, Namur, Belgium, 22 April 2005. Proceedings.
[11] Andrea Omicini and Franco Zambonelli. Coordination for Internet application development. *Autonomous Agents and Multi-Agent Systems*, 2(3):251–269, September 1999. Special Issue: Coordination Mechanisms for Web Agents.
[12] The WS-I Organization. WS-Basic Profile 1.0 document. http://www.ws-i.org.
[13] The WS-I Organization. WS-I Official web site. http://www.ws-i.org.
[14] The STIL project. The STIL official web site. http://stil.pc.unicatt.it/.
[15] Alessandro Ricci, Mirko Viroli, and Andrea Omicini. *Construenda est* CArtAgO: Toward an infrastructure for artifacts in MAS. In Robert Trappl, editor, *Cybernetics and Systems 2006*, volume 2, pages 569–574, Vienna, Austria, 18–21 April 2006. Austrian Society for Cybernetic Studies. 18th European Meeting on Cybernetics and Systems Research (EMCSR 2006), 5th International Symposium "From Agent Theory to Theory Implementation" (AT2AI-5). Proceedings.
[16] Alessandro Ricci, Mirko Viroli, and Andrea Omicini. Programming MAS with artifacts. In Rafael P. Bordini, Mehdi Dastani, Jürgen Dix, and Amal El Fallah Seghrouchni, editors, *Programming Multi-Agent Systems*, volume 3862 of *LNAI*, pages 206–221. Springer, March 2006. 3rd International Workshop (PROMAS 2005), AAMAS 2005, Utrecht, The Netherlands, 26 July 2005. Revised and Invited Papers.
[17] W3C WS Working Group. Web Services Architecture. http://www.w3.org/TR/ws-arch/.

PRACTIONIST: a Framework for Developing BDI Agent Systems

Vito Morreale*, Susanna Bonura*, Giuseppe Francaviglia*,
Michele Puccio*, Fabio Centineo*, Giuseppe Cammarata*,
Massimo Cossentino†, and Salvatore Gaglio†‡

*R&D Laboratory - ENGINEERING Ingegneria Informatica S.p.A.
†ICAR-Italian National Research Council
‡DINFO-University of Palermo

I. The framework

In this abstract we give a brief overview of the PRACTIONIST framework, which supports programmers in developing BDI agents and is built on top of JADE [1], a widespread platform that implements the FIPA[1] specifications. Therefore, our agents are deployed within JADE containers and their main cycle is implemented by means of a JADE cyclic behaviour (figure 2).

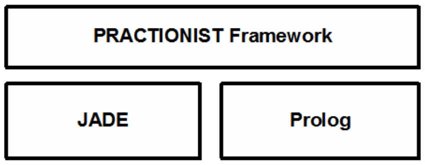


Fig. 1. PRACTIONIST over JADE and Prolog.

A PRACTIONIST agent is a software component endowed with the following elements (figure 2):

- a set of *perceptions* and the corresponding *perceptors* that listen to some relevant external stimuli;
- a set of *beliefs* representing the information the agent has got about both its internal state and the external environment;
- a set of *goals* the agent wishes or wants to pursue. They represent some states of affairs to bring about or activities to perform and will be related to either its desires or intentions (see below);
- a set of *goal relations* the agent uses during the deliberation process and means-ends reasoning;
- a set of *plans* that are the means to achieve its intentions;
- a set of *actions* the agent can perform to act over its environment; and
- a set of *effectors* that actually execute the actions.

As shown in figure 2, PRACTIONIST agents are structured in two main layers: the framework defines the execution logic and provides the built-in components according to such a logic, while the top layer includes the specific agent components to be implemented, in order to satisfy system requirements.

[1]http://www.fipa.org

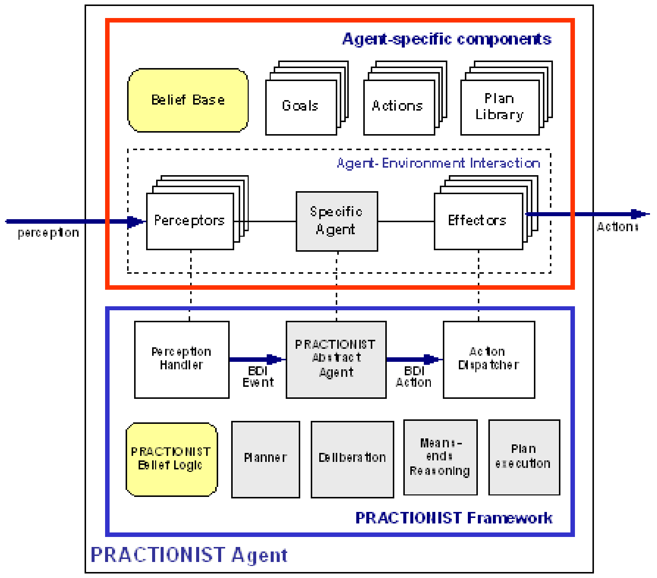


Fig. 2. Components of PRACTIONIST agents.

Therefore, a developer who wants to design an agent has to develop *(i)* the *Goals* the agent could pursue, *(ii)* the means (a set of plans, i.e. the *Plan Library*) to pursue such goals or to react to the stimuli coming from the environment, *(iii)* the *Perceptors* to receive such stimuli, *(iv)* the *Actions* the agent could perform and the corresponding *Effectors*, and *(v)* the set of beliefs and rules (*Belief Base*) to model the information about both its internal state and the external world (details on beliefs are given in [2]).

In the following section we give an overview of how to program some of agent components, with reference to the paper ”*Reasoning about Goals in BDI Agents: the PRACTIONIST Framework*”, presented at the WOA 2006 [3].

II. Implementing agent components

The concepts and the examples given in this section refer to the *tileworld* demonstrator, which is a multi agent system with two types of agents, i.e. an agent that manages the environment and player agents.

Several simulation parameters can be altered at run time, such as the appear rate and the life cycle of holes, tiles and obstacles. These information was represented by beliefs [2] about the state of the environment represented through the following predicates:

- *gridSize(width: X, height: Y)* represents the size of the grid, in terms of width and height,
- *holeBirth(rate: X)* and *holeLifecycle(rate: X)* represent the frequency of holes' birth and their mean life cycle,
- *tileBirth(rate: X)* and *tileLifecycle(rate: X)* represent the frequency of tiles' birth and their mean life cycle,
- *obstacleBirth(rate: X)* and *obstacleLifecycle(rate: X)* represent the frequency of obstacles' birth and their mean life cycle,
- *agent(name: X)* represents other active player agents.

The framework provides the support to let agent make meta-level reasoning. In other words, each player agent, by reasoning on above information, will be able to select the optimal strategy to increase its score. For example, the plan *FindTileInAmplitudePlan* implements a depth search behavior, while the plan *FindTileRandomicallyPlan* implements a random search strategy. Thus these plans are used by the player to find a tile in several circumstances.

Analogously, agent beliefs about its state refer to the following predicates:

- *position(xPos: X, yPos:Y)* represents the position of the player agent,
- *score(value: X)* represents the current score of the player,
- *hold(obj: tile)* states that the player agent holds a tile.

On the base of such beliefs, some goals are defined as well. As an example, the *HoldTile* is a state goal that succeeds when *hold(obj: tile)* is believed true by the agent for the same *tile*. Thus, in the *AchieveTilePlan*, the player agent has to identify a tile within the grid to satisfy the *HoldTile* goal and then hold such a tile by executing the action of picking it up.

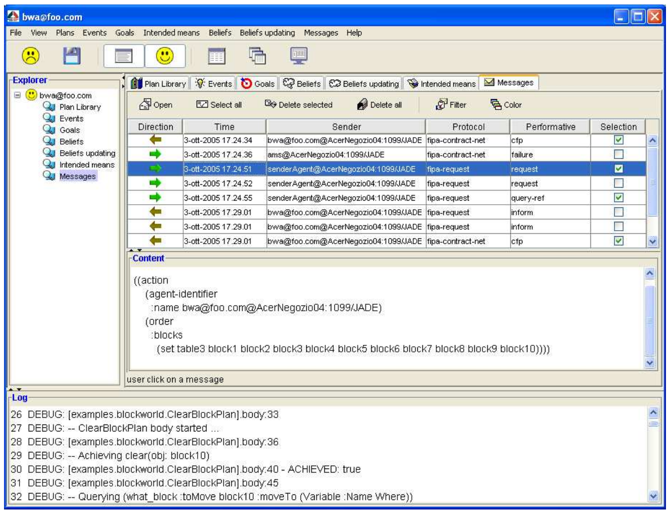
The player agent is endowed with the *Taker* effector, which triggers and executes the pick up action and updates the environment status and its internal state. The agent is also provided with other effectors (e.g. *Mover*, *Releaser*, etc.) to be able to perform other actions, such as moving itself in the grid and releasing holding tiles.

Finally, the cognitive system of the agent includes a set of perceptors that receive stimuli from the environment. As an example, the player agent is equipped with the perceptors *TileLifeCyclePerceptor*, *HoleBirthPerceptor*, etc. to be able to perceive changes from the environment about tiles' birth rate, the obstacles' life cycle, and so forth.

III. PRACTIONIST Agent Introspection Tool (PAIT)

The framework also provides developers with the PRACTIONIST Agent Introspection Tool (PAIT), a visual integrated

[2] In PRACTIONIST beliefs can be about either predicates or other beliefs (expressed by the operator Bel). Moreover, predicates can be expressed by specifying the role of their arguments, i.e. $predicate(role1 : element1, role2 : element2, ..., roleN : elementN)$.


bwa@foo.com
File View Plans Events Goals Intended means Beliefs Beliefs updating Messages Help
Explorer
bwa@foo.com
Plan Library
Events
Goals
Beliefs
Beliefs updating
Intended means
Messages
Plan Library | Events | Goals | Beliefs | Beliefs updating | Intended means | Messages
Open | Select all | Delete selected | Delete all | Filter | Color

Direction	Time	Sender	Protocol	Performative	Selection
←	3-ott-2005 17.24.34	bwa@foo.com@AcerNegozio04:1099/JADE	fipa-contract-net	cfp	✓
→	3-ott-2005 17.24.36	ams@AcerNegozio04:1099/JADE	fipa-contract-net	failure	☐
→	3-ott-2005 17.24.51	senderAgent@AcerNegozio04:1099/JADE	fipa-request	request	✓
→	3-ott-2005 17.24.52	senderAgent@AcerNegozio04:1099/JADE	fipa-request	request	☐
→	3-ott-2005 17.24.55	senderAgent@AcerNegozio04:1099/JADE	fipa-request	query-ref	✓
←	3-ott-2005 17.29.01	bwa@foo.com@AcerNegozio04:1099/JADE	fipa-request	inform	☐
←	3-ott-2005 17.29.01	bwa@foo.com@AcerNegozio04:1099/JADE	fipa-request	inform	☐
←	3-ott-2005 17.29.01	bwa@foo.com@AcerNegozio04:1099/JADE	fipa-contract-net	cfp	✓

Content
```
((action
   (agent-identifier
     :name bwa@foo.com@AcerNegozio04:1099/JADE)
   (order
     :blocks
       (set table3 block1 block2 block3 block4 block5 block6 block7 block8 block9 block10))))
```
user click on a message

Log
```
26 DEBUG: [examples.blockworld.ClearBlockPlan].body:33
27 DEBUG: -- ClearBlockPlan body started ...
28 DEBUG: [examples.blockworld.ClearBlockPlan].body:36
29 DEBUG: -- Achieving clear(obj: block10)
30 DEBUG: [examples.blockworld.ClearBlockPlan].body:40 - ACHIEVED: true
31 DEBUG: [examples.blockworld.ClearBlockPlan].body:45
32 DEBUG: -- Querying (what_block :toMove block10 :moveTo (Variable :Name Where))
```

Fig. 3. The PRACTIONIST Agent Introspection Tool (PAIT).

monitoring and debugging tool, which supports the analysis of the agent's state during its execution. In particular, the PAIT can be suitable to display, test and debug the agents' relevant entities and execution flow. Each of these components can be observed at run-time through a set of specific tabs (see figure 3); the content of each tab can be also displayed in an independent window.

All the information showed at run-time could be saved in a file, providing the programmer with the opportunity of performing an off-line analysis. Moreover, the PAIT provides an area for log messages inserted in the agent source code, according to the Log4j approach. The usage of this console and the advantages it provides are described in more details in [4].

Acknowledgments

This work is partially supported by the Italian Ministry of Education, University and Research (MIUR) through the project PASAF.

References

[1] F. Bellifemine, A. Poggi, and G. Rimassa, "JADE - a FIPA-compliant agent framework," in *Proceedings of the Practical Applications of Intelligent Agents*, 1999. [Online]. Available: http://jmvidal.cse.sc.edu/library/jade.pdf

[2] V. Morreale, S. Bonura, G. Francaviglia, M. Cossentino, and S. Gaglio, "PRACTIONIST: a new framework for BDI agents," in *Proceedings of the Third European Workshop on Multi-Agent Systems (EUMAS'05)*, 2005, p. 236.

[3] V. Morreale, S. Bonura, G. Francaviglia, F. Centineo, M. Cossentino, and S. Gaglio, "Reasoning about goals in BDI agents: the PRACTIONIST framework," in *Proceedings of Joint Workshop "From Objects to Agents"*, 2006.

[4] V. Morreale, S. Bonura, F. Centineo, A. Rossi, M. Cossentino, and S. Gaglio, "PRACTIONIST: implementing PRACTIcal reasONIng sySTems," in *Proceedings of Joint Workshop "From Objects to Agents"*, 2005.

Simulation of Minority Game in TuCSoN

Enrico Oliva
Mirko Viroli
Andrea Omicini
ALMA MATER STUDIORUM—Università di Bologna
via Venezia 52, 47023 Cesena, Italy
E-mail:{enrico.oliva,mirko.viroli,andrea.omicini}@unibo.it

I. APPLICATION DOMAIN

Minority Game (MG) is a mathematical model that takes inspiration from the "El Farol Bar" problem introduced by Brian Arthur (1). It is based on a simple scenario where at each step a set of agents perform a boolean vote which conceptually splits them in two classes: the agents in the smaller class win. In this game, a rational agent keeps track of previous votes and victories, and has the goal of winning throughout the steps of the game—for which a rational strategy has to be figured out.

One of the most important applications of MG is in the market models: (2) use MG as a coarse-grained model for financial markets to study their fluctuation phenomena and statistical properties. Even though the model is coarse-grained and provides an over-simplified micro-scale description, it anyway captures the most relevant features of system interaction, and generates collective properties that are quite similar to those of the real system.

Another point of view, presented e.g. by (3), considers the MG as a point in space of a Resource Allocation Game (RAG). In this work a generalisation of MG is presented that relaxes the constraints on the number of resources, studying how the system behaves within a given range.

MG can be considered a social simulation that aims to reproduce a simplified human scenario. In principle, a logic-based approach based on BDI agent makes it easier to explicitly model a variety of diverse social behaviours.

As showed by (4), a multiagent system (MAS) can be used to realise a MG simulation—there, BDI agents provide for rationality and planning. An agent-based simulation is particularly useful when the simulated systems include autonomous entities that are diverse, thus making it difficult to exploit the traditional framework of mathematical equations.

In order to implement MG simulations we adopt the TuCSoN infrastructure for agent coordination (5), which introduces tuple centres as artifact representatives. A tuple centre is a programmable coordination medium living in the MAS environment, used by agents interacting by exchanging tuples (logic tuples in the case of TuCSoN logic tuple centres). As we are not concerned much with the mere issues of agent intelligence, we rely here on a weak form of rationality, through logic-based agents adopting pre-compiled plans called *operating instructions* (6).

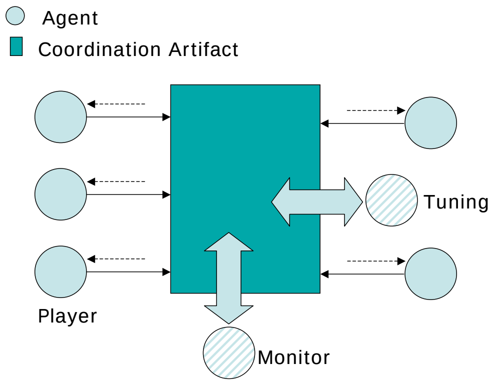


Fig. 1. TuCSoN Simulation Framework for MG

II. LOGIC ARCHITECTURE

The architecture proposed for MAS simulation is based on TuCSoN (5), which is an infrastructure for the coordination of MASs. TuCSoN provides agents with an environment made of logic tuple centres, which are logic-based programmable tuple spaces. The language used to program the coordination behaviour of tuple centres is ReSpecT, which specifies how a tuple centre has to react to an observable event (e.g. when a new tuple is inserted) and has to accordingly change the tuple-set state (7). Tuple centres are a possible incarnation of the coordination artifact notion (8), representing a device that persists independently of agent life-cycle and provides services to let agents participate to social activities.

In our simulation framework we adopt logic-based agents, namely, agents built using a logic programming style, keeping a knowledge base (KB) of facts and acting according to some rule—rules and facts thus forming a logic theory. The implementation is based on tuProlog technology[1] for Java-Prolog integration, and relies on its inference capabilities for agent rationality. Agents roughly follow the BDI architecture, as the KB models agent beliefs while rules model agent intentions.

Three kinds of agents are used in our simulation: player agents, monitor agents and tuning agents (as depicted in Figure 1): all the agents share the same coordination artifact. The agent types differ because of their role and behaviour: player agents play MG, the monitor agent is an observer

[1]http://tuprolog.alice.unibo.it

of interactions which visualises the progress of the system, the tuning agent can change some rules or parameters of coordination, and drives the simulation to new states. Note that the main advantage of allowing a dynamic tuning of parameters instead of running different simulations lays in the possibility of tackling emergent aspects which would not necessarily appear in new runs.

The main control loop of a player agent is a sequence of actions: observing the world, updating its KB, scheduling next intention, elaborating and executing a plan. To connect agent mental states with interactions we use the concept of action preconditions and perception effects as usual.

III. Minority Game Performance

To track the performance of an MG system, the most interesting quantity is *variance*, defined as $\sigma^2 = \overline{[A(t) - \overline{A(t)}]^2}$: it shows the variability of the bets around the average value $\overline{A(t)}$. In particular, the normalised version of variance $\rho = \sigma^2/N$ is considered. Figure 3 shows a typical evolution of the game.

Generally speaking, variance is the inverse of global efficiency: as variance decreases agent coordination improves, making more agents winning. Variance is interestingly affected by the parameters of the model, such as number of agents (N), memory (m) and number of strategies (s): in particular, the fluctuation of variance is shown to depend only on the ratio $\alpha = 2^m/N$ between agent memory and the number N of agents.

The results of observations suggest that the behaviour of MG can be classified in two phases: an information-rich *asymmetric* phase, and an unpredictable or *symmetric* phase. A phase transition is located where σ^2/N attains its minimum ($\alpha_c = 1/2$), and it separates the symmetric phase with $\alpha < \alpha_c$ from an asymmetric phase with $\alpha > \alpha_c$.

All these cases have been observed with the TuCSoN simulation framework described in next section.

IV. The Simulation Framework

The construction of MG simulations with MASs is based on the TuCSoN framework and on tuProlog as an inferential engine to program logic agents. The main innovative aspect of this MG simulation is the possibility of studying the evolution of the system with particular and different kinds of agent behaviour at the micro level, imposed as coordination parameters which are changed on-the-fly.

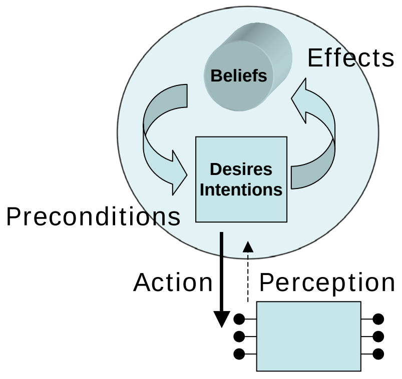


Fig. 2. Agent Architecture

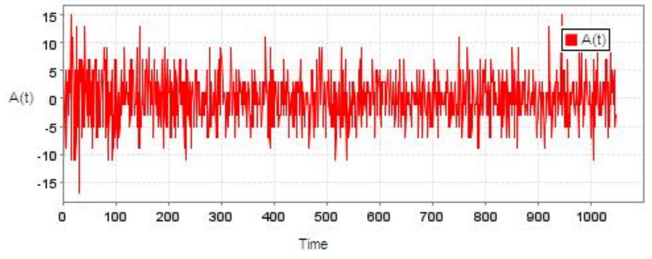


Fig. 3. Typical Time evolution of the Original MG with $N = 51$, $m = 5$ and $s = 2$

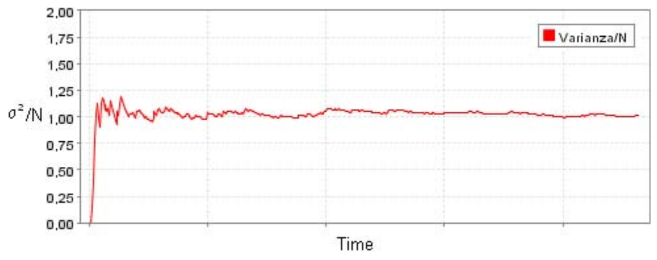


Fig. 4. Variance of the Game with 11 Random Agents

A. Operating Instructions

Each agent has an internal plan, structured as an algebraic composition of allowed actions (with their preconditions) and perceptions (with their effects), that enables the agent to use the coordination artifact to play the MG. This plan can be seen as Operating Instructions (6), a formal description based on Labelled Transition System (LTS) that the agent reads to understand what its step-by-step behaviour should be. Through an inference process, the agent accordingly chooses the next action to execute, thus performing the cycle described in Section 2.

Operating instructions are expressed by the following theory:

```
firststate(agent(first,[])).
definitions([
  def(first,[],...),
  def(main,[S],
       [act(out(play(X)),pre(choice(S,X))),
        per(in(result(Y)),eff(res(Y))),
        agent(main,[S])]
  ),
  ...
]).
```

The first part of operating instructions is expressed by term `first`, where the agent reads the game parameters that are stored in the KB, and randomly creates its own set of strategies.

In the successive part `main`, the agent executes its main cycle. It first puts tuple `play(X)` in the tuple space, where $X = \pm 1$ is agent vote. The precondition of this action `choice(S,X)` is used to bind in the KB X with the value currently chosen by the agent according to strategy S. Then, the agent gets the whole result of the game in tuple `result(Y)` and applies it to its KB. After this perception, the cycle is iterated again.

B. Tuple Centre Behaviour

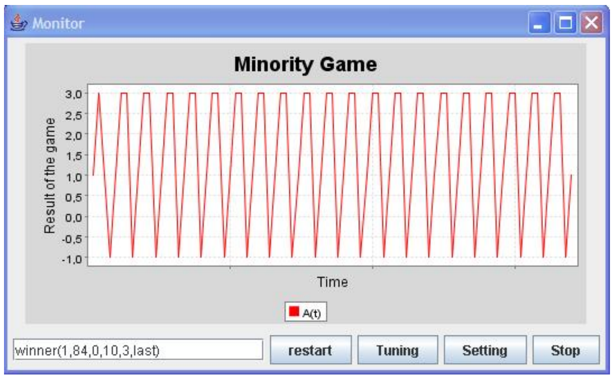
The interaction protocol between agents and the coordination artifact is then simply structured as follows. First each agent puts the tuple for its vote. When the tuples for all agents have been received, the tuple centre checks them, computes the result of the game—either 1 or -1 is winning—and prepares a result tuple to be read by agents.

The ReSpecT program for this behaviour is loaded in the tuple centre by a configuration agent at bootstrap, through operation `set_spec()`. The following ReSpecT reaction is fired when an agent inserts tuple `play(X)`, and triggers the whole behaviour:

```
reaction(out(play(X)),(
  in_r(count(Y)),
  Z is Y+1,
  in_r(sum(M)),
  V is M+X,
  out_r(sum(V)),
  out_r(count(Z))
)).
```

This reaction considers the bet (`X`) counts the bets (`Z`) and computes the partial result of the game (`V`). When all the agents have played, the artifact produces the tuple `winner(R,NS,NumberOfLoss,MemorySize,last/more)` which is the main tuple of MG coordination.

```
reaction(out_r(count(X)),(
  rd_r(numag(Num)),
  X=:=Num,
  in_r(totcount(T)),
  P is T+1,
  rd_r(game(G)),
  in_r(sum(A)),
  out_r(sum(0)),
  rd_r(countsession(CS)),
  in_r(count(Y)),
  out_r(count(0)),
  %%calculate variance
  in_r(qsum(SQ)),
  NSQ is A*A+SQ,
  out_r(qsum(NSQ)),
  %%calculate mean
  in_r(totsum(R)),
  NewS is R+A,
  out_r(totsum(NewS)),
```



winner(1,84,0,10,3,last)

Fig. 5. Interface of the Monitor Agent

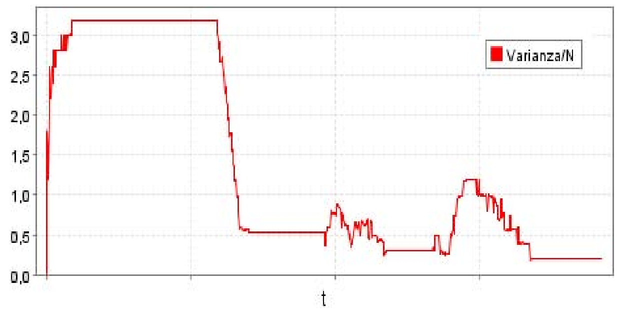


Fig. 6. Variance of the System with Initial Parameters $N = 5$ and $m = 3$

```
  rd_r(numloss(NumberOfLoss)),
  rd_r(mem(MemorySize)),
  out_r(winner(A,P,CS,NumberOfLoss,MemorySize,G)),
  out_r(totcount(P))
)).
```

The `winner` tuple contain the result of game (`R`), the number of step (`NS`), two tuning parameters (`NumberOfLoss` and `MemorySize`) and one constant to communicate agents whether they have to stop or to play further (`last/more`). Figure 5 reports the graphical interface of the monitor agent that during its life-time reads the tuple `winner` and draws variance.

C. Tuning the Simulation

In classical MG simulation there are a number of parameters that can affect the system behaviour, which are explicitly represented in the tuple centre in form of tuples: the number of agents `numag(X)`, memory size `mem(X)`, and the number of strategies `numstr(X)`. In our framework, we have introduced as a further parameter the number of wrong moves after which the single agent should be recalculate own strategy, represented as a tuple `numloss(X)`. Such a threshold is seemingly useful to break the symmetry in the strategy space when the system is in a pathological state, i.e., when all agents have the same behaviour and the game oscillates from minimum to maximum value.

In our framework, it is possible to explore the possibility to dynamically tune up the coordination rules by changing `numloss` and `mem` coordination parameters, which are stored as tuples in the coordination artifact. The simulation architecture built in this way, in fact, allows for on-the-fly change of some game configuration parameters—such as the dimension of agent memory—with no need to stop the simulation and re-program the agents.

By changing the parameters, the tuning agent can drive the system from an equilibrium state to another, by controlling agent strategies, the dimension of memory, or the number of losses that an agent can accept before discarding a strategy. This agent observes system variance, and decides whether and how to change tuning parameters: reference variance is calculated by first making agents playing the game randomly—see Figure 4. The new value of parameters is stored in tuple centre through tuples `numloss(NumberOfLoss)` and

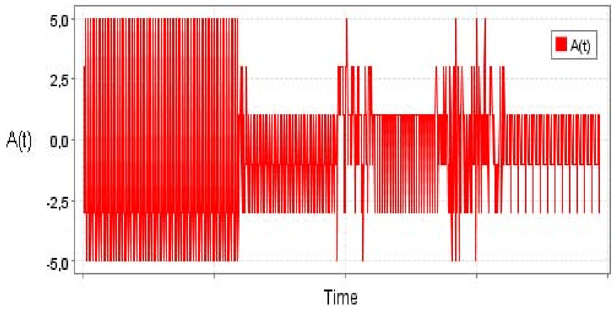


Fig. 7. System Evolution of the Variance in Figure 6

`mem(MemorySize)`, the rules of coordination react and update the information that will be read by the agents.

D. Simulation Results

The result of the tuned simulation in Figures 6 and 7 shows how the system changes its equilibrium state and achieves a better value of variance.[2] In this simulation the tuning agent is played by a human that observes the evolution of the system and acts through the tuning interface to change the coordination parameters, such as threshold of losses and memory, hopefully finding new and better configurations. The introduction of the threshold of losses in the agent behaviour is useful when the game is played by few agents: these parameters enable system evolution and a better agent cooperative behaviour.

V. Perspectives

In this paper, we aim at introducing new perspectives on agent-based simulation by adopting a novel MAS meta-model based on agents and artifacts, and by applying it to Minority Game simulation. We implement and study MG over the TuCSoN coordination infrastructure, and show some benefits of the artifact model in terms of flexibility and controllability of the simulation. In particular, in this work we focus on the possibility to build a feedback loop on the rules of coordination driving a system to a new and better equilibrium state.

We foresee some new perspectives in the use of the TuCSoN simulation framework in a industrial environment. The first one is to use the system to drive manufacturing in case of limited resources. In this scenario each agent is a half-processed item, whose production has to be completed as faster as possible, and whose access to the resources is regulated by dedicated resource artifacts. Another possible perspective is to evaluate the product demand and production in order to drive industry through market fluctuation. In our framework we could model a market scenario by minority rules, and then try to evaluate demand. Furthermore, all such applications would benefit from using a logic-based approach rather than an equation-based approach.

[2]In Figure 6, the first phase of equilibrium is followed by a second one obtained by changing the threshold parameter $S = 5$. Finally, a third phase is obtained changing the dimension of the memory to $m = 5$.

VI. Acknowledgements

The first author of this paper would like to thank Dr. Peter McBurney and the Department of Computer Science at University of Liverpool for their scientific support and their hospitality during his stay in Liverpool, when this paper was mostly written.

References

[1] W. B. Arthur, "Inductive reasoning and bounded rationality (the El Farol problem)," *American Economic Review*, vol. 84, no. 2, pp. 406–411, May 1994.

[2] D. Challet, M. Marsili, and Y.-C. Zhang, "Modeling market mechanism with minority game," *Physica A: Statistical and Theoretical Physics*, vol. 276, no. 1–2, pp. 284–315, Feb. 2000. [Online]. Available: http://dx.doi.org/10.1016/S0378-4371(99)00446-X

[3] H. V. D. Parunak, S. Brueckner, J. Sauter, and R. Savit, "Effort profiles in multi-agent resource allocation," pp. 248–255.

[4] W. Renz and J. Sudeikat, "Modeling Minority Games with BDI agents – a case study," in *Multiagent System Technologies*, ser. LNCS, T. Eymann, F. Klügl, W. Lamersdorf, M. Klusch, and M. N. Huhns, Eds. Springer, 2005, vol. 3550, pp. 71–81, 3rd German Conference (MATES 2005), Koblenz, Germany, 11-13 Sept. 2005. Proceedings. [Online]. Available: http://www.springerlink.com/link.asp?id=y62q174g56788gh8

[5] A. Omicini and F. Zambonelli, "Coordination for Internet application development," *Autonomous Agents and Multi-Agent Systems*, vol. 2, no. 3, pp. 251–269, Sept. 1999.

[6] M. Viroli and A. Ricci, "Instructions-based semantics of agent mediated interaction," in *3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*, N. R. Jennings, C. Sierra, L. Sonenberg, and M. Tambe, Eds., vol. 1. New York, USA: ACM, 19–23 July 2004, pp. 102–109. [Online]. Available: http://portal.acm.org/citation.cfm?id=1018409.1018737

[7] A. Omicini and E. Denti, "Formal ReSpecT," *Electronic Notes in Theoretical Computer Science*, vol. 48, pp. 179–196, June 2001.

[8] A. Omicini, A. Ricci, M. Viroli, C. Castelfranchi, and L. Tummolini, "Coordination artifacts: Environment-based coordination for intelligent agents," in *3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*, N. R. Jennings, C. Sierra, L. Sonenberg, and M. Tambe, Eds., vol. 1. New York, USA: ACM, 19–23 July 2004, pp. 286–293. [Online]. Available: http://portal.acm.org/citation.cfm?id=1018409.1018752

[9] N. R. Jennings, C. Sierra, L. Sonenberg, and M. Tambe, Eds., *3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*. New York, USA: ACM, 19–23 July 2004.

Software Agents for Learning Nash Equilibria in Non-Cooperative Games

Alfredo Garro, Marco Iusi

Abstract—**This paper describes SALENE, a Multi Agent System (MAS) for learning Nash Equilibria in non-cooperative games. SALENE is based on the following assumptions: if agents representing the players act as rational players, i.e. they act to maximise their expected utility in each match of a game, and if such agents play k matches of the game they will converge in playing one of the Nash Equilibria of the game. SALENE can be conceived as a heuristic and efficient method to compute at least one Nash Equilibria in a non-cooperative game represented in its normal form.**

Index Terms—**Multi-Agent Systems, Game Theory, Nash Equilibria.**

I. Introduction

The complexity of NASH [21], the problem consisting in computing Nash equilibria in non-cooperative games, is considered one of the most important open problem in Complexity Theory [22]. In 2005, Daskalakis, Goldbergy, and Papadimitriou showed that the problem of computing a Nash equilibrium in a game with four or more players is complete for the complexity class PPAD[1] [7], moreover, Chen and Deng extended this result for 2-player games [5]. However, even in the two players case, the best algorithm known has an exponential worst-case running time [23]; furthermore, if the computation of equilibria with simple additional properties is required, the problem immediately becomes NP-hard [3, 6, 11, 12].

Motivated by these results, recent studies have dealt with the problem of computing Nash Equilibria by exploiting approaches based on the concepts of learning and evolution [10, 15]. In these approaches the Nash Equilibria of a game are not statically computed but are the result of the evolution of a system composed by agents playing the game. In particular, each agent after different rounds will learn to play a strategy that, under the hypothesis of agent's rationality, will be one of the Nash equilibria of the game [2, 4, 9, 13, 18].

In this paper we present SALENE, a MAS for learning Nash Equilibria in non-cooperative games. In particular, given a static non cooperative game described in its normal form, the agents of the system will play the static game k times; after each match each agent will decide which strategy to play in the next match on the basis of his beliefs about the strategies that the other agents are adopting. More specifically, each agent assumes that his beliefs about the other players' strategies are correct and he plays a strategy that is a best response to his beliefs. By increasing k the agents will converge in playing one of the Nash equilibria of the game.

This paper is structured as follows. In Section 2, a formal definition of the problem will be given and the system requirements detailed. In Section 3 and in Section 4 the design and the implementation of SALENE will be described respectively, then, in Section 5, some experimental results will be shown. Finally, in Section 6, conclusions and future efforts will be addressed.

II. Problem Definition and System Requirements

An n-person strategic game G can be defined as a tuple $G = (N; (A^i)_{i\in N}; (r^i)_{i\in N})$, where $N = \{1, 2, \dots, n\}$ is the set of players, A^i is a finite set of actions for player $i\in N$, and $r^i : A^1 \times \dots \times A^n \to \Re$ is the payoff function of player i. The set A^i is called also the set of pure strategies of player i. The Cartesian product $\times_{i\in N}A^i = A^1 \times \dots \times A^n$ can be denoted by A and $r : A \to \Re^N$ can denote the vector valued function whose ith component is r^i, i.e., $r(a) = (r^1(a), \dots, r^n(a))$, so it is possible to write (N, A, r) for short for $(N; (A^i)_{i\in N}; (r^i)_{i\in N})$.

For any finite set A^i the set of all probability distributions on A^i can be denoted by $\Delta(A^i)$. An element $\sigma^i \in \Delta(A^i)$ is a mixed strategy for player i.

A (Nash) equilibrium of a strategic game $G = (N, A, r)$ is an N-tuple of (mixed) strategies $\sigma = (\sigma^i)_{i\in N}$, $\sigma^i \in \Delta(A^i)$, such that for every $i \in N$ and any other strategy of player i, $\tau^i \in \Delta(A^i)$, $r^i(\tau^i,\sigma^{-i}) \le r^i(\sigma^i,\sigma^{-i})$, where r^i denotes also the expected payoff to player i in the mixed extension of the game and σ^{-i} represents the mixed strategies in σ of all the other players. Basically, supposing that all the other players do not change their strategies it is not possible for any player i to play a different strategy τ^i able to gain a better payoff of that gained by playing σ^i. σ^i is called a Nash equilibrium strategy for player i.

In 1951 J. F. Nash proved that a strategic (non-cooperative) game $G = (N, A, r)$ has a (Nash) equilibrium σ [17]; in his honour, the computational problem of finding such equilibria is known as NASH [21].

A. Garro is with the Department of Electronics, Informatics and Systems (DEIS), University of Calabria, Rende (CS), 87036 Italy. (e-mail: alfredo.garro @unical.it).

M. Iusi is with the Department of Electronics, Informatics and Systems (DEIS), University of Calabria, Rende (CS), 87036 Italy. (e-mail: marco.iusi@tin.it).

[1] PPAD (polynomial parity argument, directed version) class was introduced by Papadimitriou in his seminal work in 1991 [20].

In order to exemplify the definitions given above let us consider a game with two players (n=2) and $|A^1|=|A^2|=m$, i.e., the sets of pure strategies have both cardinality equals to m [3]. In this case the set of pure strategies for each player could be identified with the ordered set M = {1, 2, . . . , m} and the game could be represented by two $m \times m$ matrices B and W. The first player is called the row player and the second player is called the column player. If the row player plays strategy i and the column player strategy j, the payoff will be B_{ij} for the first player and W_{ij} for the second player.

A mixed strategy, a probability distribution over pure strategies, is a vector $\chi \in \Re^M$ such that $\sum_{s \in M} \chi_s = 1$ and for every $s \in$ M, $\chi_s \geq 0$.

When the row player plays mixed strategy χ and the column player plays mixed strategy γ, their expected payoffs will be, respectively, $\chi^t B \gamma$ and $\chi^t W \gamma$ (χ^t is the transpose of vector χ).

A Nash Equilibrium of the game described by the matrices B and W is a pair of mixed strategies (χ, γ) such that for all mixed strategies $\overline{\chi}$ and $\overline{\gamma}$, of the row and the column player respectively, $\chi^t B \gamma \geq \overline{\chi}^t B \gamma$ and $\chi^t W \gamma \geq \chi^t W \overline{\gamma}$.

Starting from the problem definition discussed above, SALENE was conceived as a system for learning at least one Nash Equilibrium of a non-cooperative game given in the form $G = (N; (A^i)_{i \in N}; (r^i)_{i \in N})$. In particular, the system asks the user for:

- the number n of the players which defines the set of players $N = \{1, 2, \dots, n\}$;
- for each player $i \in N$, the related finite set of pure strategies A^i and his payoff function $r^i : A^1 \times \dots \times A^n \rightarrow \Re$;
- the number k of times the players will play the game.

Then, the system creates n agents, one associated to each player, and a referee. The players and the referee both know that G is the actual game to be played, i.e. there is *complete information* [8, 19]. Each player is a rational player i.e. his goal is to maximise his expected utility/payoff[2]. In particular, in SALENE a rational player acts to maximise his expected utility in each single match without considering the overall utility that he could obtain in a set of matches.

This kind of agents will play the game G k times, after each match, each agent will decide the strategy to play in the next match to maximise his expected utility on the basis of his beliefs about the strategies that the other agents are adopting. By increasing k the agents will converge in playing one of the Nash Equilibria of the game. This conclusion relays on the hypothesis that the agents will act as rational players and derives straightly from the assumptions on which the Nash's theorem is based [8, 17, 19, 25].

[2] Payoffs are numeric representations of the utility obtainable by a player in the different outcomes of a game.

III. System Design

On the basis of the requirements highlighted in the previous section the SALENE (Software Agent for LEarning Nash Equilibria) MAS was designed. The class diagram of SALENE is shown in Figure 1.

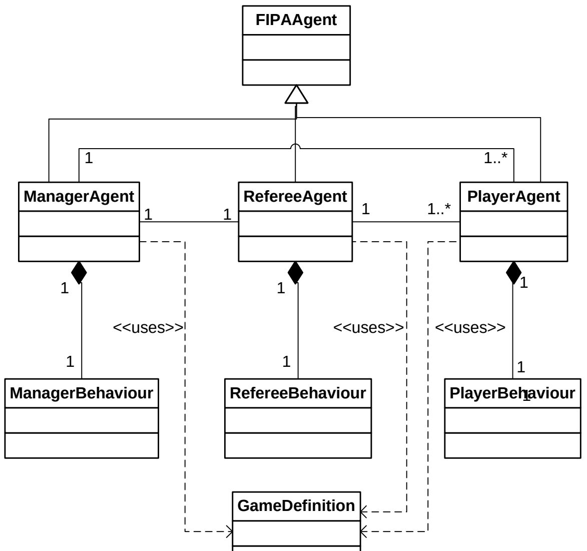


Fig. 1. Class diagram of SALENE

The Manager Agent interacts with the user and it is responsible for the global behaviour of the system. In particular, after having obtained from the user the input parameters G and k (see section II), the Manager Agent creates both n Player Agents, one associated to each player, and a Referee Agent that coordinates and monitors the behaviours of the players. The Manager Agent sends to all the agents the definition G of the game then he asks the Referee Agent to orchestrate k matches of the game G. In each match, the Referee Agent asks each Player Agent which pure strategy he has decided to play, then, after having acquired the strategies from all players, the Referee Agent communicates to each Player Agent both the strategies played and the payoffs gained by all players. After playing k matches of the game G the Referee Agent communicates all the matches' data to the Manager Agent which analyses it and properly presents the obtained results to the user.

A Player Agent is a rational player that, given the game definition G, acts to maximise his expected utility in each single match of G. In particular the behaviour of the Player Agent i can be described by the following main steps:

1. In the first match the Player Agent i chooses to play a pure strategy randomly generated considering all the pure strategies playable with the same probability: if $|A^i|=m$ the probability of choosing a pure strategy $s \in A^i$ is $1/m$.
2. The Player Agent i waits for the Referee Agent to ask him which strategy he wants to play, then he communicates to the Referee Agent the chosen pure strategy as computed in step 1 if he is playing his first match or in step 4 otherwise;

3. The Player Agent waits for the Referee Agent to communicate him both the pure strategies played and the payoffs gained by all players;
4. The Player Agent decides the mixed strategy to play in the next match. In particular, the Player Agent updates the beliefs about the mixed strategies currently adopted by the other players and consequently recalculate the strategy able to maximise his expected utility. Basically, the Player Agent i tries to find the strategy $\sigma^i \in \Delta(A^i)$, such that for any other strategy $\tau^i \in \Delta(A^i)$, $r^i(\tau^i,\sigma^{-i}) \leq r^i(\sigma^i,\sigma^{-i})$ where r^i denotes his expected payoff and σ^{-i} represents his beliefs about the mixed strategies currently adopted by all the other players, i.e. $\sigma^{-i}=(\sigma^j)_{j\in N, j\neq i}$, $\sigma^j \in \Delta(A^j)$. In order to evaluate σ^j for each other player $j\neq i$ the Player Agent i considers the pure strategies played by the player j in all the previous matches and computes the frequency of each pure strategy, this frequency distribution will be the estimate for σ^j. If there is at least an element in the actually computed set $\sigma^{-i}=(\sigma^j)_{j\in N, j\neq i}$ that differs from the set σ^{-i} as computed in the previous match, the Player Agent i solves the inequality $r^i(\tau^i,\sigma^{-i}) \leq r^i(\sigma^i,\sigma^{-i})$ that is equivalent to solve the optimization problem $P=\{\max(r^i(\sigma^i,\sigma^{-i})), \sigma^i\in\Delta(A^i)\}$. It is worth noting that P is a linear optimization problem, actually, given the set σ^{-i}, $r^i(\sigma^i,\sigma^{-i})$ is a linear objective function in σ^i (see the two players example reported in Section II), and with $|A^i|=m$ $\sigma^i\in\Delta(A^i)$ is a vector $\chi \in \Re^M$ such that $\sum_{s\in M} \chi_s = 1$ and for every $s\in M$ $\chi_s\geq 0$, so the constraint $\sigma^i\in\Delta(A^i)$ is a set of $m+1$ linear inequalities. P is solved by the Player Agent by using an efficient method for solving problems in linear programming [14, 16], in particular the predictor-corrector method of Mehrotra [16], whose complexity is polynomial for both average and worst case. The obtained solution for σ^i is a pure strategy because it is one of the vertices of the polytope which defines the feasibility region for P. The obtained strategy σ^i will be played by the Player Agent i in the next match; $r^i(\sigma^i,\sigma^{-i})$ represents the expected payoff to player i in the next match;
5. back to step 2.

The Manager Agent, receives from the Referee Agent all the data about the k matches of the game G and computes an estimate of a Nash Equilibrium of G, i.e. an N-tuple $\sigma=(\sigma^i)_{i\in N}$, $\sigma^i\in\Delta(A^i)$. In particular, in order to estimate σ^i (the Nash equilibrium strategy of the player i), the Manager Agent computes, on the basis of the pure strategies played by the player i in each of the k match, the frequency of each pure strategy: this frequency distribution will be the estimate for σ^i. The so computed set $\sigma=(\sigma^i)_{i\in N}$, $\sigma^i\in\Delta(A^i)$ will be then properly proposed to the user together with the data exploited for its estimation.

IV. System Implementation

The JADE-based classes of SALENE were straightforwardly derived from the class diagram reported in Figure 1. In particular:

– *ManagerAgent*, *RefereeAgent* and *PlayerAgent* extend the *Agent* class of JADE [1];
– *ManagerBehaviour*,*RefereeBehaviour* and *PlayerBehaviour* extend *FSMBehaviour* class of JADE which models a complex task whose sub-tasks correspond to the activities performed in the states of a finite state machine. In particular, the behaviours of both the Referee and the Player Agent are also cyclic.

The interactions among SALENE Agents are appositely defined through sequences of ACL messages instances of the *ACLMessage* class of JADE.

V. Experimental Results

SALENE was tested on different games that differ from each other both in the number and in the kind of Nash Equilibria. This section presents the results obtained for three popular games: (1) *The Prisoner's Dilemma* which has one pure Nash Equilibrium (that is an equilibrium in which all the players play a pure strategy); (2) *Matching Pennies* which has one mixed Nash Equilibrium (that is an equilibrium in which at least one player plays a mixed strategy); (2) *Battle of the Sexes* which has three Nash Equilibria (two Pure Equilibria and one Mixed Equilibrium).

A. The Prisoner's Dilemma

An informal description of the *Prisoner's Dilemma* can be found in [24]. Formally, in a game G of *Prisoner's Dilemma* (PD), two players (n=2) simultaneously choose a move, either cooperate (c) or defect (d), so $A^1=A^2=\{c,d\}$ and $|A^1|=|A^2|=m=2$. There are thus four possible outcomes for each encounter: both cooperate (cc), the first player cooperates, while the second defects (cd), vice versa (dc), and both players defect (dd). Each player receives a payoff after each encounter as reported in Table Ia-b. Table I semantic derives straightly from the *bimatrix* representation of a two-player game as discussed in Section II. In particular, the move of Player 1 determines the row, the move of Player 2 determines the column, and the pair (X,Y) in the corresponding cell indicates that payoff of Player 1 is X and the payoff of Player 2 is Y. Regarding the payoffs reported in Table Ia the following order must hold: T>R>P>L. Table Ib shows a valid assignment for the payoffs.

TABLE I
(A) Payoffs For Prisoner's Dilemma

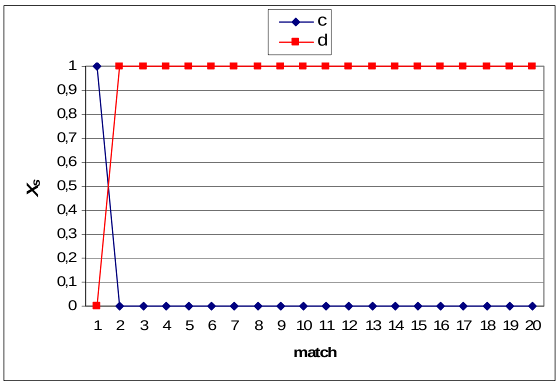
		Player 2	
		c	d
Player 1	c	R,R	L,T
	d	T,L	P,P

(B) A Valid Assignment for the Payoffs (T>R>P>L)

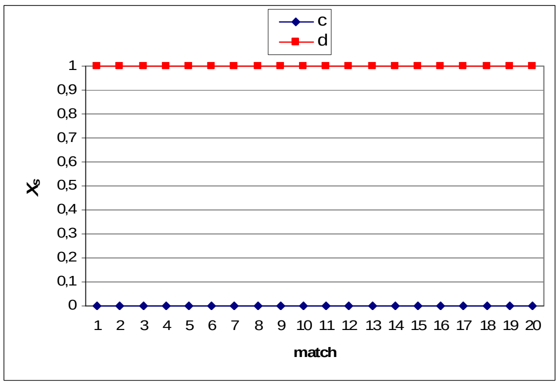
		Player 2	
		c	d
Player 1	c	6,6	0,10
	d	10,0	3,3

By looking at Table Ib, it is possible to note that for Player 1 d is the best response to c (10>6) and d is also the best response to d (3>0). The same is true for Player 2, so both players rationally will play their pure strategy d that is their dominant strategy. Formally, the *Prisoner's Dilemma* has one Nash Equilibrium $\sigma=\{\sigma^1,\sigma^2\}=\{\chi,\gamma\}$, $\sigma^1\in\Delta(A^1)$, $\sigma^2\in\Delta(A^2)$, $\chi\in\Re^2$, $\gamma\in\Re^2$, where $\chi=[0, 1]$ and $\gamma=[0, 1]$.

In order to compare the analytical result with the result obtainable in SALENE we ran 30 experiments each consisting of 100 matches (k=100) of the *Prisoner's Dilemma*. In the case of the *Prisoner's Dilemma* the expected result was that as soon as a Player Agent played his dominant strategy *d*, he would never change his choice, so after few matches the Player Agents played both their dominant strategy *d* so converging in playing the Nash Equilibrium of the game. The experiments confirm the expected result, as an example Figure 2a-b reports one of the experiments carried out. In particular, Figure 2a(2b) shows the strategy played by Player 1(Player 2) in each of the *k* match of an experiment. As showed in Figure 2a-b, after few matches both the Player Agents play their pure strategy *d* as required by the Nash Equilibrium of the game.



(a) Strategy played by Player 1



(b) Strategy played by Player 2

Fig. 2. The Prisoner's Dilemma: experimental results

B. Matching Pennies

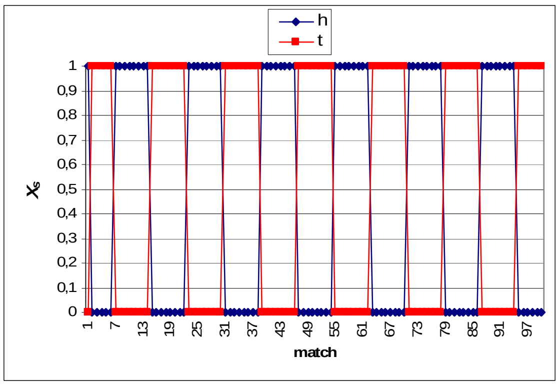
An informal description of the *Matching Pennies* game follows: the game is played between two players, each player has a penny and must secretly turn it to heads or tails, the players then reveal their choices simultaneously; if the pennies match (both heads or both tails), Player 1 receives S dollars from Player 2. If the pennies do not match (one heads and one tails), Player 2 receives S dollars from Player 1. This is an example of a zero-sum game, where one player's gain is exactly equal to the other player's loss. Formally, in a game *G* of *Matching Pennies* (MP), two players (n=2) simultaneously choose a move, either heads (h) or tails (t), so $A^1=A^2=\{h,t\}$ and $|A^1|=|A^2|=m=2$. Each player receives a payoff after each encounter as reported in Table IIa-b.

TABLE II
(A) PAYOFFS FOR MATCHING PENNIES

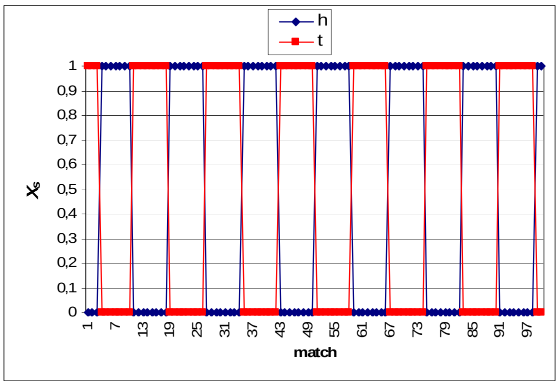
		Player 2	
		h	t
Player 1	h	L,-L	-L,L
	t	-L,L	L,-L

(B) A VALID ASSIGNMENT FOR THE PAYOFFS (S=1)

		Player 2	
		h	t
Player 1	h	1,-1	-1,1
	t	-1,1	1,-1



(a) Strategy played by Player 1



(b) Strategy played by Player 2

Fig. 3. Matching Pennies: experimental results

Matching Pennies has no pure strategy Nash equilibrium since there is no pure strategy (heads or tails) that is a best response to a best response, i.e. a dominant strategy. Alternatively, there is not a pure strategy that a player would ever change when told the pure strategy played by the other player. Instead, the unique Nash equilibrium of *Matching Pennies* is in mixed strategies: each player chooses heads or tails with equal probability. In this way, each player makes the other indifferent in choosing heads or tails, so neither player has an incentive to try another strategy. Formally, *Matching Pennies* has one mixed Nash Equilibrium $\sigma=\{\sigma^1,\sigma^2\}=\{\chi,\gamma\}$, $\sigma^1\in\Delta(A^1)$, $\sigma^2\in\Delta(A^2)$, $\chi\in\Re^2$, $\gamma\in\Re^2$, where $\chi=[0.5, 0.5]$ and $\gamma=[0.5, 0.5]$.

In order to compare the analytical result with the result obtainable in SALENE we ran 30 experiments each consisting of 100 matches (k=100) of *Matching Pennies*. The expected result was that, analyzing the pure strategies played by each player in each of the k match, their frequency distribution would asymptotically converge to the mixed Nash Equilibrium of the game. The experiments confirm the expected result: by increasing k the computed frequency distributions asymptotically converge to the mixed Nash Equilibria of the game. As an example Figure 3a-b reports one of the experiments carried out. In this case the computed frequency distributions were: $\sigma^1=\chi=[0.49, 0.51]$ and $\sigma^2=\gamma=[0.49, 0.51]$.

C. *Battle of Sexes*

An informal description of the *Battle of Sexes* game follows: a man and a woman plan to meet after work to attend an event: an opera or a football match, but they can not communicate so they have to choose separately where to go. The woman prefers the opera to the football match, whereas the man prefers the football match to the opera, but both prefer to be together at either event than alone at either one. More formally in a game G of *Battle of Sexes* (BS), two players (n=2) simultaneously choose a move, either opera (o) or football (f), so $A^1=A^2=\{o,f\}$ and $|A^1|=|A^2|=m=2$. Each player receives a payoff after each encounter as reported in Table IIIa-b. Regarding the payoffs reported in Table IIIa the following order must hold: T>R>L. Table IIIb shows a valid assignment for the payoffs.

TABLE III
(A) Payoffs For Battle of Sexes

		Player 2	
		o	f
Player 1	o	T,R	L,L
	f	L,L	R,T

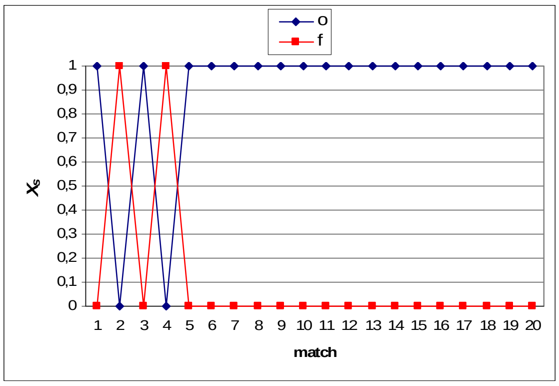
(B) A Valid Assignment for the Payoffs (T>R>L)

		Player 2	
		o	f
Player 1	o	3,2	0,0
	f	0,0	2,3

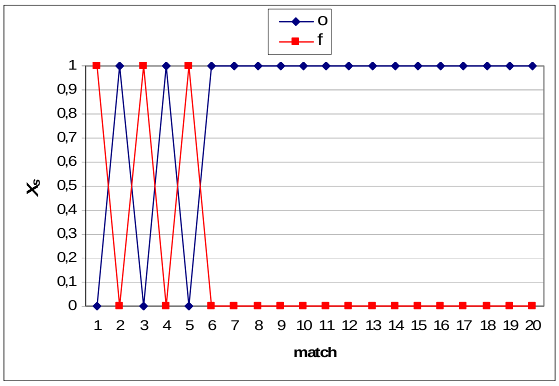
Battle of Sexes has two pure strategy Nash equilibria, one where both go to the opera and another where both go to the football game; there is also a Nash equilibrium in mixed strategies, where, given the payoffs listed in Table IIIb, each player attends their preferred event with probability 2/3. Formally, *Battle of Sexes* has three Nash Equilibria:
$\sigma_I=\{\sigma^1_I,\sigma^2_I\}=\{\chi_I,\gamma_I\}$, $\sigma^1_I \in \Delta(A^1)$, $\sigma^2_I \in \Delta(A^2)$, $\chi_I \in \Re^2$, $\gamma_I \in \Re^2$, where $\chi_I=[1, 0]$ and $\gamma_I=[1, 0]$;
$\sigma_{II}=\{\sigma^1_{II},\sigma^2_{II}\}=\{\chi_{II},\gamma_{II}\}$, $\sigma^1_{II} \in \Delta(A^1)$, $\sigma^2_{II} \in \Delta(A^2)$, $\chi_{II} \in \Re^2$, $\gamma_{II} \in \Re^2$, where $\chi_{II}=[0, 1]$ and $\gamma_{II}=[0, 1]$;
$\sigma_{III}=\{\sigma^1_{III},\sigma^2_{III}\}=\{\chi_{III},\gamma_{III}\}$, $\sigma^1_{III} \in \Delta(A^1)$, $\sigma^2_{III} \in \Delta(A^2)$, $\chi_{III} \in \Re^2$, $\gamma_{III} \in \Re^2$, where $\chi_{III}=[2/3, 1/3]$ and $\gamma_{III}=[1/3, 2/3]$;

In order to compare the analytical result with the result obtainable in SALENE, we ran 30 experiments each consisting of 100 matches (k=100) of the *Battle of Sexes*. *Battle of Sexes* presents an interesting case for games theory since each of the Nash Equilibria is deficient in some way. The two pure strategy Nash Equilibria are unfair, one player consistently does better than the other. In the mixed strategy Nash Equilibrium the players will be together at the same event with probability 4/9 and will be alone with probability 5/9, leaving each player with an expected payoff of 10/9 that is very low if compared with the expected payoff of the two pure Nash Equilibria.

The expected result was that as soon as both the Player Agents played the same pure strategy (o or f), i.e. one of the Pure Nash Equilibria of the game, the Agents would never change their choices: the Player who plays his favorite strategy will not have incentive to change it, the player who does not play his favorite strategy will not change it because in this case his expected payoff will get worse in the next match. In particular, after 1 or h*(T+R) matches, $k \geq h \geq 1$, there is a probability of 50% that from this match on the Player Agents will converge in playing one of the Pure Nash Equilibria of the game.



(a) Strategy played by Player 1



(b) Strategy played by Player 2
Fig. 4. Battle of Sexes: experimental results

The experiments confirm the expected result: in all the experiments after 1 or h*(T+R) matches the Player Agents play one of the two pure Nash Equilibria of the game. As an example Figure 4a-b reports one of the experiments carried out, in this case the played Nash Equilibrium was σ_I.

VI. Conclusions

The complexity of NASH, the problem consisting in computing Nash equilibria in non-cooperative games, is still debated, but even in the two players case, the best algorithm known has an exponential worst-case running time. Starting from these considerations SALENE, a MAS for learning Nash Equilibria in non cooperative games, was developed.

SALENE is based on the assumptions that if agents representing the players act as rational players, i.e. if each player acts to maximise his expected utility in each match of a game *G*, and if such agents play *k* matches of *G* they will converge in playing one of the Nash Equilibria of the game. In particular, after each match each agent decides the strategy to play in the next match on the basis of his beliefs about the strategies that the other agents are adopting. More specifically, each agent assumes that his beliefs about the other players' strategies are correct and plays a strategy that is a best response to his beliefs. Analyzing the behaviour of each agent in all the *k* matches of *G*, SALENE presents to the user an estimate of a Nash Equilibrium of the game.

A set of experiments was carried out on different games that differ from each other both in the number and in the kind of Nash Equilibria. The experiments demonstrated that:

- if the game has one Pure Nash Equilibrium the agents converge in playing this equilibrium;
- if the game has one Mixed Nash Equilibrium, the frequency distributions of the pure strategies played by each player asymptotically converge to the mixed Nash Equilibrium of the game;
- if the game has $p>1$ Pure Nash Equilibria and $s>1$ Mixed Nash Equilibria the agents converge in playing one of the *p* Pure Nash Equilibria.

SALENE can be conceived as a heuristic and efficient method for computing at least one Nash Equilibria in a non-cooperative game represented in its normal form; actually, the learning algorithm adopted by the Player Agents has a polynomial running time [14, 16] for both average and worst case.

Efforts are currently underway to: (i) evaluate different learning algorithms and extensively testing them on complex games; (ii) let the user ask for the computation of equilibria with simple additional properties.

REFERENCES

[1] F. Bellifemine, A. Poggi, and G. Rimassa. *Developing multi-agent systems with a FIPA-compliant agent framework*. In Software Practice and Experience, 31, pp. 103-128, 2001.

[2] M. Benaim and M.W. Hirsch. *Learning process, mixed equilibria and dynamic system arising for repeated games*. In Games Econ. Behav., 29, pp. 36-72, 1999.

[3] V. Bonifaci, U. Di Iorio, and L. Laura. *On the complexity of uniformly mixed Nash equilibria and related regular subgraph problems*. In Proceedings of the 15th International Symposium on Fundamentals of Computation Theory (FCT) 2005.

[4] D. Carmel and S. Markovitch. *Learning Models of Intelligent Agents*. In Proceedings of the 13th National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference, Vol. 2, pp 62-67, AAAI Press, Menlo Park, California, 1996.

[5] X. Chen and X. Deng. *Settling the Complexity of 2-Player Nash-Equilibrium*. Electronic Colloquium on Computational Complexity, Report No. 140 (2005).

[6] V. Conitzer and T. Sandholm. *Complexity results about Nash equilibria*. In Proceedings of the 18th Int. Joint Conf. on Artificial Intelligence, pages 765–771, 2003.

[7] C. Daskalakis, P. W. Goldbergy, and C. H. Papadimitriou. *The Complexity of Computing a Nash Equilibrium*. Electronic Colloquium on Computational Complexity, Report No. 115 (2005).

[8] A.K. Dixit and S. Skeath. *Games of Strategy (2nd edition)*. W.W. Norton & Company, 2004.

[9] D.B. Fogel. *Evolving Behaviours in the Iterated Prisoner's Dilemma*, Evolutionary Computation, Vol. 1:1, pp 77-97, 1993.

[10] D. Fudenberg and D. Levine. *The Theory of Learning in Games*. Cambridge, MA, MIT Press, 1998.

[11] I. Gilboa and E. Zemel. *Nash and correlated equilibria: some complexity considerations*. Games and Economic Behavior, 1(1):80–93, 1989.

[12] G. Gottlob, G. Greco, and F. Scarcello. *Pure Nash Equilibria: hard and easy games*. In Proceedings of the 9th Conference on Theorethical Aspects of Rationality and Knowledge (TARK-2003), Bloomington, Indiana, USA, June 20-22, 2003.

[13] P. Jehiel and D. Samet. *Learning to Play Games in Extensive Form by Valuation*, NAJ Economics, Peer Reviews of Economics Publications, 3, 2001.

[14] N. Karmarkar. *A New Polynomial-time Algorithm for Linear Programming*. In Combinatorica 4, 373-395, 1984.

[15] J. Maynard Smith. *Evolution and the Theory of Games*. Cambridge University Press, 1982.

[16] S. Mehrotra. *On the Implementation of a Primal-dual Interior Point Method*. In SIAM J. Optimization 2, 575-601, 1992.

[17] J. F. Nash. *Non-cooperative games*. In Annals of Mathematics, volume 54, pages 289–295, 1951.

[18] A. Neyman. *Finitely Repeated Games With Finite Automata*. In Mathematics Of Operations Research, Vol. 23, No. 3, August 1998.

[19] M. J. Osborne. *An Introduction to Game Theory*. Oxford University Press, 2002.

[20] C. H. Papadimitriou. *On inefficient proofs of existence and complexity classes*. In Proceedings of the 4th Czechoslovakian Symposium on Combinatorics, 1991.

[21] C. H. Papadimitriou. *On the complexity of the parity argument and other inefficient proofs of existence*. Journal of Computer and Systems Sciences, 48(3):498–532, 1994.

[22] C. H. Papadimitriou. *Algorithms, Games and the Internet*. In Proceedings of the 33rd Annual ACM Symposium on the Theory of Computing (STOC), pages 749–753, 2001.

[23] R. Savani and B. von Stengel. *Exponentially many steps for finding a Nash equilibrium in a bimatrix game*. In Proceedings of the 45th Symp. Foundations of Computer Science, pages 258–267, 2004.

[24] G. Tsebelis. *Nested Games: rational choice in comparative politics*. University of California Press, 1990.

[25] J. Von Neumann and O. Morgenstern. *Theory of Games and economic Behaviour*, Princeton University Press, 1944.

A multidimensional flocking algorithm for clustering spatial data

Antonio Augimeri, Gianluigi Folino, Agostino Forestiero and Giandomenico Spezzano
Institute for High Performance Computing and Networking (ICAR-CNR)
augimeriantonio@email.it, (folino, forestiero, spezzano)@icar.cnr.it

Abstract—**In this paper, we describe the efficient implementation of M-Sparrow, an adaptive flocking algorithm based on the biology-inspired paradigm of a flock of birds. We extended the classical flock model of Reynolds with two new characteristics: the movement in a multi-dimensional space and different kinds of birds. The birds, in this context, are used to discovery point having some desired characteristics in a multidimensional space. A critical point of the algorithm is the efficient search of the k-neighbors in a multidimensional space. This search was efficiently implemented using the ANN libraries.**

I. Introduction

Ants'colonies, flocks of birds, termites, swarms of bees etc. are agent-based insect models that exhibit a collective intelligent behavior (*swarm intelligence*) [2] that may be used to define new **distributed clustering algorithms**. In these models, the emergent collective behavior is the outcome of a process of self-organization, in which insects are engaged through their repeated actions and interaction with their evolving environment. Intelligent behavior frequently arises through indirect communication between the agents using the principle of **stigmergy** [6]. This mechanism is a powerful principle of cooperation in insect societies. According to this principle an agent deposits something in the environment that makes no direct contribution to the task being undertaken but it is used to influence the subsequent behavior that is task related. Swarm intelligence (SI) models have many features in common with Evolutionary Algorithms (EA). Like EA, SI models are population-based. The system is initialized with a population of individuals (i.e., potential solutions). These individuals are then manipulated over many iteration steps by mimicking the social behavior of insects or animals, in an effort to find the optima in the problem space. Unlike EAs, SI models do not explicitly use evolutionary operators such as crossover and mutation. A potential solution simply 'flies' through the search space by modifying itself according to its past experience and its relationship with other individuals in the population and the environment.

These algorithms show a high level of robustness to change by allowing the solution to dynamically adapt itself to global changes by letting the agents self-adapt to the associated local changes.

In this paper, we present a prototype using a new algorithm based on the concepts of a flock of birds that move together in a complex manner with simple local rules, to explore multidimensional spaces for searching interesting objects. The algorithm is an extension of the classical flock model of Reynolds with two new characteristics: the movement in a multi-dimensional space and different kinds of birds. The birds, in this context, are used to discovery point having some desired characteristics in a multidimensional space. The implementation is based on SWARM [7], a software package for multi-agent simulation of complex systems, developed at the Santa Fe Institute.

From an efficiency point of view the most critical point of the algorithm is the efficient search of the k-neighbors (i.e. the cardinality of the neighborhood) in a multidimensional space. This search was efficiently implemented using the ANN (Approximate Nearest Neighbor) libraries.

The remainder of this paper is organized as follows. Section 2 describes the adaptive flocking algorithm, section 3 shows how the approach was applied to multidimensional spaces. Section 4 shows some interesting experimental results about the efficiency of the algorithm and its efficacy in finding interesting patterns. Finally section 5 draws some conclusions.

II. The adaptive flocking algorithm

The classical flocking model, introduced by Reynolds [8], moves in a two dimensional space and all the birds have the same characteristics. In the next subsections, we describe the two extensions introduced in M-Sparrow, the use of birds having different characteristics (represented by different colors) and the movement in a multidimensional space.

A. *Reynolds' original flock model*

The flocking algorithm was originally proposed by Reynolds as a method for mimicking the flocking behavior of birds on a computer both for animation and as a way to study emergent behavior. Flocking is an example of emergent collective behavior: there is no leader, i.e., no global control. Flocking behavior emerges from the local interactions. Each agent has direct access to the geometric description of the whole scene, but reacts only to flock mates within a certain small radius. The basic flocking model consists of three simple steering behaviors: separation, cohesion and alignment.

Separation gives an agent the ability to maintain a certain distance from others nearby. This prevents agents from crowding too closely together, allowing them to scan a wider area. Cohesion gives an agent the ability to cohere (approach and form a group) with other nearby agents. Steering for cohesion can be computed by finding all agents in the local neighborhood and computing the average position of the nearby agents. The steering force is then applied in the direction of that average position. Alignment gives an agent the ability to align with other nearby characters. Steering for alignment can be computed by finding all agents in the local neighborhood and averaging together the 'heading' vectors of the nearby agents.

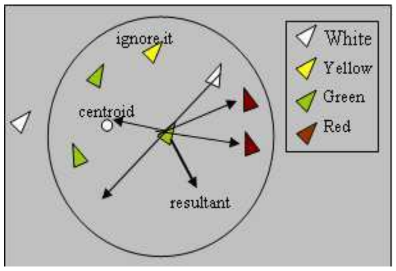


Fig. 1. Computing the direction of a green agent.

B. *An adaptive colored flocking algorithm*

M-SPARROW extends the Reynolds' flocking algorithm, described in the previous subsection, considering four different kinds of agents, classified on the basis of some properties of data in their neighborhood. These different kinds are characterized by a different color: red, revealing interesting patterns in the data, green, a medium one, yellow, a low one, and white, indicating a total absence of patterns. In practise, the flock follows an exploring behavior in which individual members (agents) to first explore the environment searching for goals whose positions are not known *a priori*, and then, after the goals are located, all the flock members should move towards these goals. Agents search the goals in parallel and signal the presence or the lack of significant patterns into the data to other flock members, by changing color. The main idea behind our approach is to take advantage of the colored agent in order to explore more accurately the most interesting regions (signaled by the red agents) and avoid the ones without clusters (signaled by the white agents). Red and white agents stop moving in order to signal this type of regions to the others, while green and yellow ones fly to find more dense clusters. Indeed, each flying agent computes its heading by taking the weighted average of alignment, separation and cohesion (as illustrated in figure 1). The entire flock then moves towards the agents (*attractors*) that have discovered interesting regions to help them, avoiding the uninteresting areas that are instead marked as obstacles. The color is assigned to the agents by a function associated with the data analyzed. In practice, the agent computes the property of the explored point and then it chooses the color (and the speed) in accordance to the simple rules showed in table I.

So *red*, reveals a high density of interesting patterns in the data, *green*, a medium one, *yellow*, a low one, and white, indicates a total absence of patterns. The color is used as a communication mechanism among flock members to indicate them the roadmap to follow. The roadmap is adaptively adjusted as the agents change their color moving to explore data until they reach the goal.

$property > threshold$	$\Rightarrow$	$mycolor = red$	$(speed = 0)$
$\frac{threshold}{4} < property \leq threshold$	$\Rightarrow$	$mycolor = green$	$(speed = 1)$
$0 < property \leq \frac{threshold}{4}$	$\Rightarrow$	$mycolor = yellow$	$(speed = 2)$
$property = 0$	$\Rightarrow$	$mycolor = white$	$(speed = 0)$

TABLE I

Assigning speed and color to the agents

Green and yellow agents compute their movement observing the positions of all other agents that are at most at some fixed distance (*dist_max*) from them and applying the rules of Reynolds' [8] with the following modifications:

- *Alignment* and *cohesion* do not consider yellow agents, since they move in a not very attractive zone.
- *Cohesion* is the resultant of the heading towards the average position of the green flockmates (centroid), of the attraction towards red agents, and of the repulsion by white agents.
- A *separation* distance is maintained from all the agents, whatever their color is.

Agents will move towards the computed destination with a speed depending from their color: green agents will move more slowly than yellow agents since they will explore denser zones of clusters. An agent will speed up to leave an empty or uninteresting region whereas it will slow down to investigate an interesting region more carefully. The variable speed introduces an adaptive behavior in the algorithm. In fact, agents adapt their movement and change their behavior (speed) on the basis of their previous experience represented from the red and white agents.

```
for i=1 ... MaxIterations
    foreach agent (yellow, green)
        age=age+1;
        if (age > Max_Life)
            generate_new_agent();die();
        endif
        if (not visited (current_point))
            property = compute_property(current_point);
            mycolor= color_agent(property);
        endif
    end foreach
    foreach agent (yellow, green)
        dir= compute_dir();
    end foreach
    foreach agent (all)
        switch (mycolor){
            case yellow, green: move(dir, speed(mycolor)); break;
            case white: stop(); generate_new_agent(); break;
            case red: stop(); generate_new_close_agent(); break; }
    end foreach
end for
```

Fig. 2. The pseudo-code of M-SPARROW.

During simulations a cage effect, was observed; in fact, some agents could remain trapped inside regions surrounded by red or white agents and would have no way to go out, wasting useful resources for the exploration. So, a limit on their life was imposed to avoid this effect; hence, when their age exceeded a determined value (maxLife) they were killed and were regenerated in a new randomly chosen position of the space.

C. Exploring multidimensional spaces

We wanted use our adaptive flocking algorithm in order to explore multidimensional space for searching point having desired properties. A continuous data point can be represented in a multidimensional Euclidean space, simply normalizing its attributes. Our algorithm can search for any kind of properties. In particular, we describe a useful property for the task of clustering. Given a radius (*Eps*) and a minimum number (*MinPts*) of points. A core point is a point with at least MinPts number of points in an Eps-neighborhood of the itself. Searching points having these characteristic could be useful for different task (i.e. db-scan [3] use these points to perform the task of clustering databases). In the experimental section, we will show the evaluation of our algorithm in effectively cope with this task.

In the following, we give a more formal description of the extension of the flocking algorithm to the multidimensional space. Consider a multidimensional space with dimension d. Each bird k can be represented as a point in this space, having coordinates $x_{k1}, x_{k2}, \ldots, x_{kd}$ and having direction $\theta_{k1}, \theta_{k2}, \ldots, \theta_{kd}$, where θ_{ki} represent the angle between the new direction of the bird k (computed using the rules of the previous subsection) and the axis i. Each bird moves following the the rules of the previous subsection having speed v_k. Then, for each iteration t, the new position of the bird k can be computed as:

$$\forall i = 1 \ldots d \qquad x_{ki}(t+1) = x_{ki}(t) + v_k \times c_{ki} \quad (1)$$

where c_{ki} represents the projection along the i axis of the direction of the boid k. Note that each components is obtained summing the respective three components of alignment, separation and cohesion (i.e. $c_{ki} = c\_alignment_{ki} + c\_separation_{ki} + c\_cohesion_{ki}$). In a multidimensional space, we can compute the components as:

$$c_{k1} = \prod_{j=1}^{d-1} cos(\theta_{kj})$$
$$c_{ki} = sin(\theta_{ki-1}) \prod_{j=i}^{d-1} cos(\theta_{kj}) \qquad i = 2 \ldots d \quad (2)$$

these formulas can be computed as a generalization of the three-dimensional case illustrated in figure 3.

From a computational point of view, a critical point of our algorithm is the efficient search of the k-neighbors in a multidimensional space. Computing exact nearest neighbors can be very expensive when dimension increases.

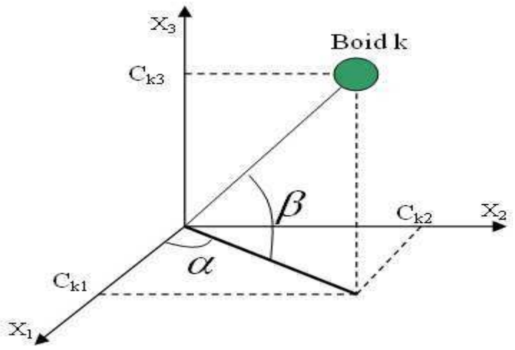


Fig. 3. Computing the components of the boid in a three-dimensional space.

ANN (Approximate Nearest Neighbor) [1] is a library written in C++, which supports data structures and algorithms for both exact and approximate nearest neighbor searching in high dimensional spaces. As showed in figure 4, we integrated ANN libraries using Java Native Interface with M-Sparrow in order to efficiently compute the neighbors necessary to our algorithm.

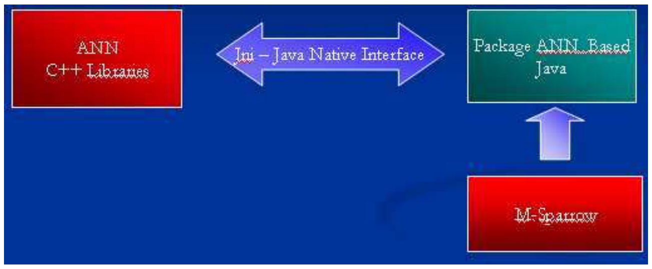


Fig. 4. Integrating M-Sparrow and ANN libraries.

III. Experimental results

In two previous papers, we demonstrated the goodness of our algorithm in discovering clusters with different sizes, shapes in noise data [5] and also with different densities [4] in a two dimensional space.

Now, we want to show how the algorithm works in a multidimensional space. We have built two dataset, each one constituted by two gaussian distributions with different densities of 1500 points. The first was three dimensional and the latter four dimensional. We run our algorithm using 200 birds for 600 iterations with k = 50 and radius = 20. It succeeds in separating almost perfectly the two gaussian distributions in both the cases. The three dimensional case is illustrated in figure 5.

Furthermore, we want to verify the efficiency of the ANN based implementation and then we run our algorithm comparing execution times of the latter ANN-based version with the previous that used brute force computation for searching the k-neighbors. The results of this comparison for a three dimensional case are reported in figures 6 a and b. Experiments show that the ANN libraries outperforms the brute force approach in all the cases and the difference is really considerable when the number of neighbors to search is greater than 50. If we consider datasets with dimension larger than 3, ANN outperforms the brute force approach by at least two order of magnitude, also for small values of k.

IV. Conclusions

We have presented an efficient implementation of an adaptive flocking algorithm that efficiently search multidimensional spaces. The implementation is based on the ANN libraries performing an efficient search of the k neighbors. Experiments showed that the algorithm is able to separate clusters in multidimensional spaces and it outperforms the previous brute force approach in terms of execution time.

References

[1] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of ACM*, 45(6):891–923, 1998.

[2] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. Swarm intelligence: From natural to artificial systems. *J. Artificial Societies and Social Simulation*, 4(1), 2001.

[3] Martin Ester, Hans-Peter Kriegel, Jorg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining*, pages 226–231, 1996.

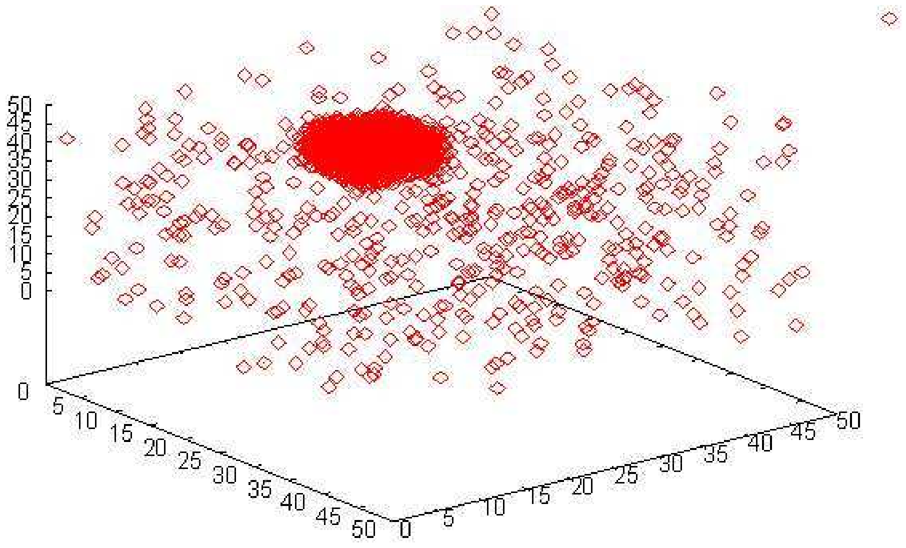
[4] Gianluigi Folino, Agostino Forestiero, and Giandomenico Spezzano. Swarming agents for discovering clusters in spatial data. *ispdc*, 2003.

[5] Gianluigi Folino and Giandomenico Spezzano. An adaptive flocking algorithm for spatial clustering. In *PPSN*, pages 924–933, 2002.

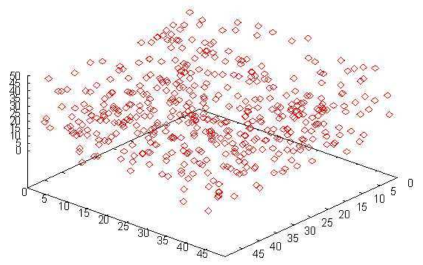
[6] P.P. Grass. *La Reconstruction du nid et les Coordinations Inter-Individuelles chez Beellicositermes Natalensis et Cubitermes sp. La Thorie de la Stigmergie : Essai d'interprtation du Comportement des Termites Constructeurs in Insect. Soc. 6.* Morgan Kaufmann, 1959.

[7] N. Minar, R. Burkhart, C. Langton, and M. Askenazi. The swarm simulation system, a toolkit for building multi-agent simulations, 1996.

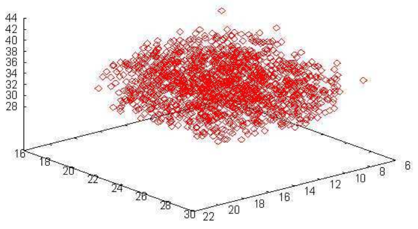
[8] Craig W. Reynolds. Flocks, herds and schools: A distributed behavioral model. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 25–34, New York, NY, USA, 1987. ACM Press.



(a)

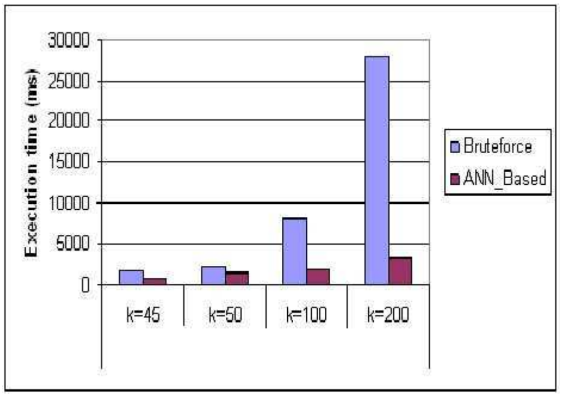


(b)

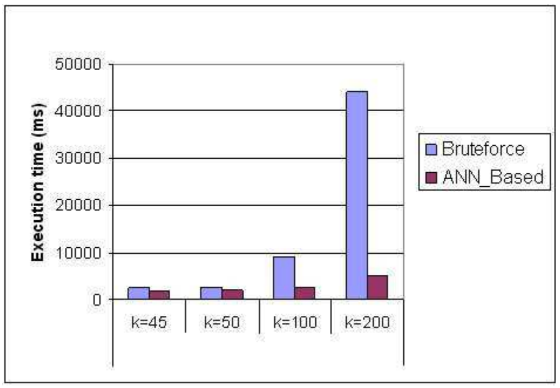


(c)

Fig. 5. Gaussian dataset. a) Original Gaussian dataset used in the experiments; b) First cluster extracted by the algorithm; c) Second cluster extracted by the algorithm.



(a)



(b)

Fig. 6. Execution time of the searching phase for the ANN-based version vs the Brute-force approach: a)Radius = 20; b) radius = 50.

MetaMeth
A Tool For Process Definition And Execution

Roberto Caico [2], Massimo Cossentino [2,3], Luca Sabatucci [1], Valeria Seidita [1], Salvatore Gaglio [1,2]

(1) DINFO - Dipartimento di Ingegneria Informatica, Università degli Studi di Palermo - Viale delle Scienze, 90128 Palermo, Italy
(2) Istituto di Calcolo delle Reti ad Alte Prestazioni, Consiglio Nazionale delle Ricerche;
(3) SET - Université de Technologie Belfort-Montbéliard - 90010 Belfort cedex, France
robertocaico@libero.it; sabatucci@csai.unipa.it; cossentino@pa.icar.cnr.it; gaglio@unipa.it

I. Introduction

Nowadays, several different tools are used in Software Engineering; in this work we are mainly interested to those supporting the design phases. These are usually classified in three categories: CASE, CAME, CAPE tools. MetaMeth is a CAME and a CAPE tool at the same time.

A CAME tool is a computerized tool that supports the method engineer in the construction of its own methods that is principally based on reuse so it aids in storing the reusable part of existing methodologies (*method fragments*) and in providing the interface for a useful and easy retrieval and assembly of fragments. MetaMeth allows a method engineer to interface with a repository of method fragment in order to retrieve them for creating his own methodology.

A CASE tool as an automated tool devotes to help the designer in the software development process by providing a software support for a reliable development of activities, lowering the risk of errors and enhancing the productivity; this definition even implies activities like planning, and management, administrative and technical aspects of a project. MetaMeth allows to manage the process through the workflow engine, the agents devoted to design activities and the expert system.

II. MetaMeth – System Requirements

Our work started with the identification of the requirements for the tool to be built. The first part of these requirements refer to the CAME functionalities:

1. **Fragment Repository**: the tool supports a fragment repository collecting methods coming from several design processes; each fragment is an extension of an existing repository we built according to the work done within FIPA, extended with the proper set of expert system rules and software components (agents) used to support the specific GUIs required in the fragment; the repository of fragments already exists and we are working for integrating it in the tool. We also considered important to plan an easy extensibility of our tool supporting several different design processes, but at the moment only the fragments used to implement our own methodologies (classic and agile process) are already fully integrated in the tool.
2. **Process Definition**. Starting from the repository of method fragments the tool allows the composition of new processes. We decided to adopt a standard by OMG, the Software Process Engineering Metamodel (SPEM) for modeling our methodologies. Once the process is modelled in SPEM we use a graphical tool (JaWE) to produce its XPDL translation (XPDL is the process specification language adopted by WFMC for describing workflow processes).

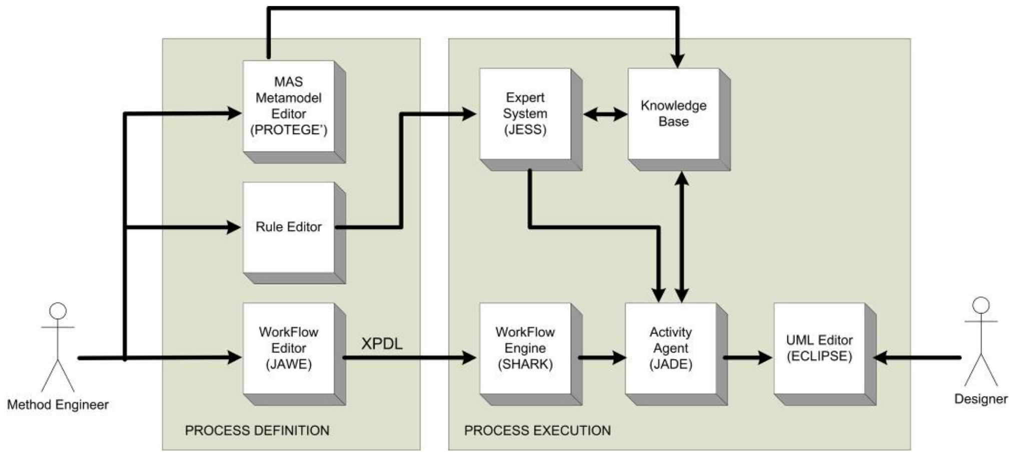


Figure 1 – Architecture of the MetaMeth application

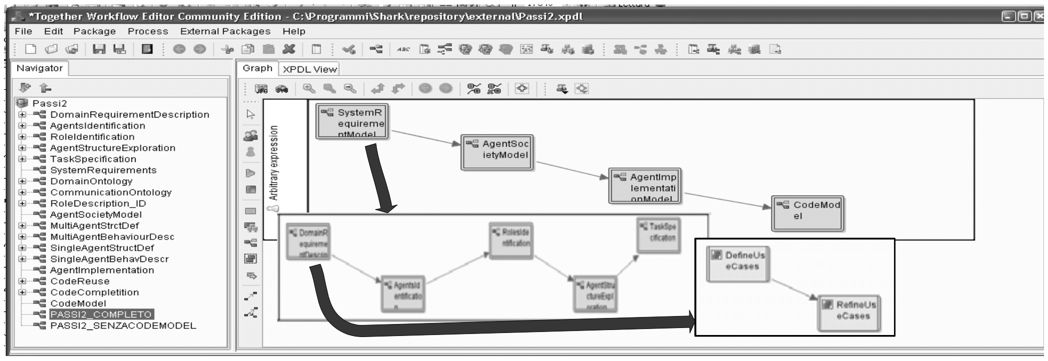


Figure 2 - A screenshot of the JaWE tool used for design the PASSI methodology

3. **Process Lifecycle.** The tool should support iterative and incremental design processes composed of several activities (eventually organized in some kind of hierarchy) with several different possible iteration paths.
4. **Notation, Syntactic and Semantic Rules**. In a methodology almost all work products have to respect three types of rules: notational, syntactic and semantic. The CAME tool have to allow the introduction of syntactic and semantic rules (expressed in a first order logic language) and to assign a graphical notation among available for working with fragments coming from different methodologies (a set of editors, developed ad-hoc for this scope, will be available).
5. **Process Roles**. During the composition of the method the method engineer may assign activities to perform to different human roles involved in the process.

The second list of requirements is related to the CASE functionalities:

1. **Process Execution**. The first requirement of this section is to instantiate a methodology built using the MetaMeth/CAME (received as XPDL specifications and a set of rules for notation, semantic and syntactic verification) and to orchestrate all the CASE services in order to design a system.
2. **Team Work**. Our tool support distributed design processes (involving several designer working on different phases at the same moment in different locations)
3. **Automatic Composition**. The tool keep in consideration dependencies among work products and include the possibility of automatically compose all (or portion of) diagrams that allow such an help.
4. **Automatic Verification**. Syntax check on notational aspects of the project and the consistency check on some design aspects like the correct instantiation of the most important elements of the MAS meta-model
5. **Reuse and Code Generation**. The tool integrates a reuse technique based on design patterns; these are collected in a repository and may be used during design. The tool has a code generator that uses an MDA approach to transform the design view in a implementation view and finally in a code view; using this approach different coding languages or implementation frameworks could be adopted. The tool supports also the production of an adequate documentation that, starting from the work products, is able of creating complex documents merging diagrams, text, tables and so on.

III. Software Architecture and Tecnologies

The prototype we developed is based on several technologies, standards and existing tools, that are independent software for enacting the process definition and the process execution; this is shown in Figure 1.

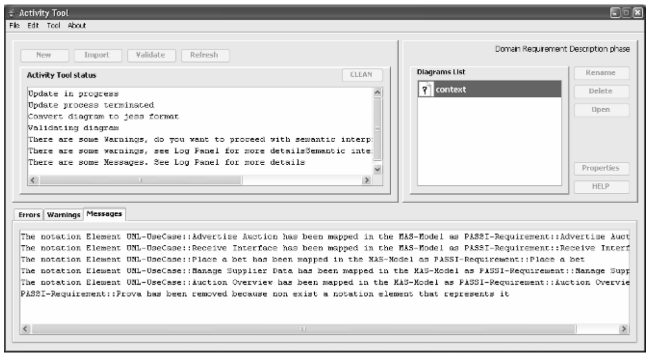
The most relevant open source components we reused in our Metameth tool are:

1. JaWE (by Enhydra) adopted as a graphical workflow editor to design the process (it exports the process using the XPDL format);
2. Shark (again by Enhydra) adopted as a workflow execution engine (it is able of reading XPDL files and also to interact with our Java-based Activity Agents); this tool ensures the design process instantiation and allows the distributed and asynchronous execution of the different activities.
3. Jade is the platform we used to develop our Activity Agents; this is the most diffused FIPA-compliant agent development platform.
4. Jess is Java-based rule engine we used to build our expert system; such a system is the ‘intelligent’ part of the Metameth tool; a relevant portion of its services are required by the Activity Agents that need reasoning capabilities in order to assist the designer in his duty.
5. The MAS meta-model required by the adopted methodology is depicted in form of an ontology using the tool Protégé by the Stanford University, California.
6. IBM Eclipse is the IDE we used to develop our UML editors.

IV. Example.

Now we are going to illustrate with an example the main steps of the construction of a new methodology and its enactment with the MetaMeth tool.

The scenario starts with JaWE session used to define the new methodology; this tool offers a graphical interface to model the process as a flow of sub-processes, and activities; each activity may be atomic or be decomposed in sub-activities. In Figure 2 we report a screenshot of the tool showing three different boxes each one related to a piece of the methodology at a different level of abstraction. The first box (the top one)


Activity Tool

Domain Requirement Description phase

Activity Tool status

```
Update in progress
Update process terminated
Convert diagram to jess format
Validating diagram
There are some Warnings, do you want to proceed with semantic interp
There are some warnings, see Log Panel for more detailsSemantic inte
There are some Messages. See Log Panel for more details
```

Diagrams List: context

Messages

```
The notation Element UML-UseCase::Advertise Auction has been mapped in the MAS-Model as PASSI-Requirement::Advertise Auct
The notation Element UML-UseCase::Receive Interface has been mapped in the MAS-Model as PASSI-Requirement::Receive Interf
The notation Element UML-UseCase::Place a bet has been mapped in the MAS-Model as PASSI-Requirement::Place a bet
The notation Element UML-UseCase::Manage Supplier Data has been mapped in the MAS-Model as PASSI-Requirement::Manage Supp
The notation Element UML-UseCase::Auction Overview has been mapped in the MAS-Model as PASSI-Requirement::Auction Overvie
PASSI-Requirement::Prova has been removed because non exist a notation element that represents it
```

Figure 3 - An example of activity tool for the Domain Requirement Description phase of PASSI

describes the main phases of our methodology (*system requirements*, *agent society*, *agent implementation* and *code model*). In the second box there is an exploitation of the system requirement phase in its composing activities (*domain description*, *agent identification*, *role description*, *agent structure exploration* and *task specification*); finally, in the third box (the lowest one), the *domain description* activity is decomposed in two atomic operations (*define use case* and *refine use case*).

The next step in the definition of the methodology is the specification in terms of Jess rules of the semantic of the MAS meta-model elements and the (work products) composition rules of the process. This is done using Protégé to draw the ontology and a Rule editor tool we built to describe Jess rules starting from some templates.

When the methodology has been entirely described using the XPDL language (process aspects) and Jess rules (semantics and composition rules), the process administrator may instantiate it using the process execution module. This has been developed using a multi-agent system composed by:

1. A Controller agent (that is interfaced with the workflow execution engine).
2. One or more Stakeholder agents (one for each designer that uses it to accept, start, decline the activities assigned to him from the process definition). After a log in session, the user may verify his activity list and can start/ refuse or delegate an activity.
3. An Expert System agent (used to wrap the Jess engine).
4. One or more Activity agents.

When the designer chooses of performing an activity, an agent becomes responsible for coordinating all the operations related to the specific activity (also in collaboration with the Expert agent and the UML editors). Activity agents offer several services to the designer: i) auto-composition used when a work product can be automatically modified/created or updated; ii) notation interpretation, used to map notational elements (use cases, classes, activities, …) into elements of the MAS meta model (requirements, agents, behaviours, …), iii) semantic validation used to verify the semantic consistence of the whole project.

Figure 3 shows the user interface of the Activity agent associated to the domain requirement description phase of our methodology; semantic interpretation and validation have been already done with the result that use cases have been mapped to requirements. In Figure 4 the first three work products of the methodology are reported (domain requirements description, agent identification and role identification).

V. Future Works

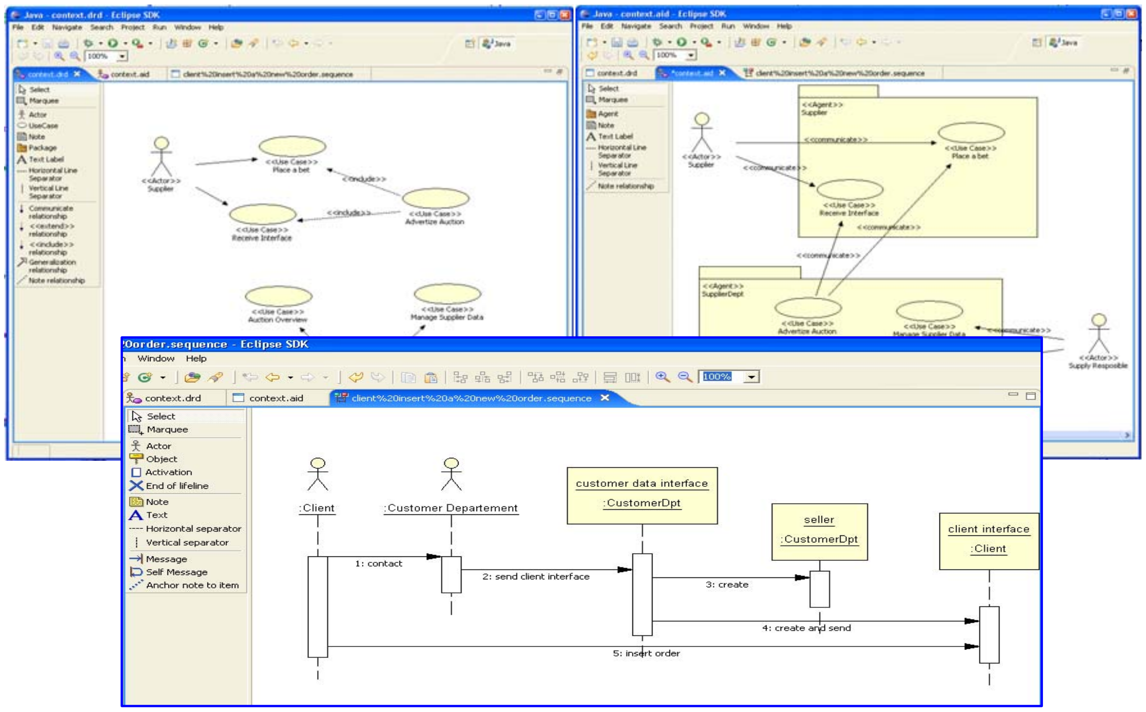


Figure 4 - Some screenshots of the UML editor developed as a plug-in for Eclipse

In the future we are going to complete this tool with more features, specifically we are going to interface it with an agent-oriented pattern reuse tool that also allows code generation for one of the most diffused agent development platforms (Jade). The production of an extensive and well-formatted documentation from the design artifact is also scheduled and will be obtained through a society of agents, each one specialized for the composition of one specific kind of document.

Another improvement, we are working on, is the population of the fragment repository, extracting methods from the existing agent-oriented methodologies.

Model driven design and implementation of activity-based applications in Hermes

Ezio Bartocci*, Flavio Corradini*, Emanuela Merelli*, Leonardo Vito*
* Dipartimento di Matematica e Informatica,
Università di Camerino,
Via Madonna delle Carceri, 62032 Camerino, Italy
{*name.surname*}@unicam.it

***Abstract*— Hermes is an agent-based middleware structured as a component-based and 3-layered software architecture. Hermes provides an integrated, flexible programming environment for design and execution of activity-based applications in distributed environments. By using workflow technology, it supports even a non expert user programmer in the model driven design and implementation of a domain specific application. In this paper, after a description of Hermes software architecture, we provide a simple demo in biological domain and we show some real case studies in which Hermes has been validated.**

I. Introduction

Hermes [9] is an agent-based middleware, for design and execution of activity-based applications in distributed environments. It supports mobile computation as an application implementation strategy. While middleware for mobile computing has typically been developed to support physical and logical mobility, Hermes provides an integrated environment where application domain experts can focus on designing activity workflow and ignore the topological structure of the distributed environment. Generating mobile agents from a workflow specification is the responsibility of a context-aware compiler. Agents can also developed directly by an expert user using directly the Application Programming Interface (API) provided by Hermes middleware. The Hermes middleware layer, compilers, libraries, services and other developed tools together result in a very general programming environment, which has been validated in two quite disparate application domains, one in industrial control [6] and the other in bioinformatics [13]. In the industrial control domain, embedded systems with scarce computational resources control product lines. Mobile agents are used to trace products and support self-healing. In the bionformatics domain, mobile agents are used to support data collection and service discovery, and to simulate biological system through autonomous components interactions. This paper is organized as follows. Section II describes the Hermes Software Architecture. Section III provides a simple demo in biological domain. In Section IV, we present several projects in which Hermes middleware has been adopted. We conclude in Section V.

II. Hermes Software Architecture

Hermes is structured as a component-based, agent-oriented system with a 3-layer software architecture shown in Figure 1: user layer, system layer and run-time layer. At the user layer, it allows designers to specify their application as a workflow of activities using the graphical notation. At the system layer, it provides a context-aware compiler to generate a pool of user mobile agents from the workflow specification. At the run-time layer, it supports the activation of a set of specialized service agents, and it provides all necessary components to support agent mobility and communication. The main difference between the run-time layer and the system layer is how agents function in each. ServiceAgents in the run-time layer are localized to one platform to interface with the local execution environment. UserAgents in the system layer are workflow executors, created for a specific goal that, in theory, can be reached in a finite time by interacting with other agents. Afterwards that agent dies.

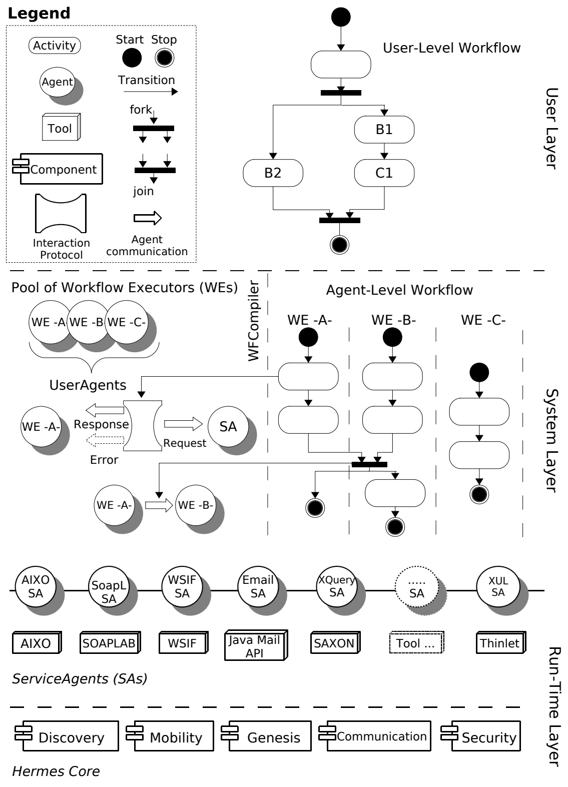


Fig. 1. Hermes Software Architecture

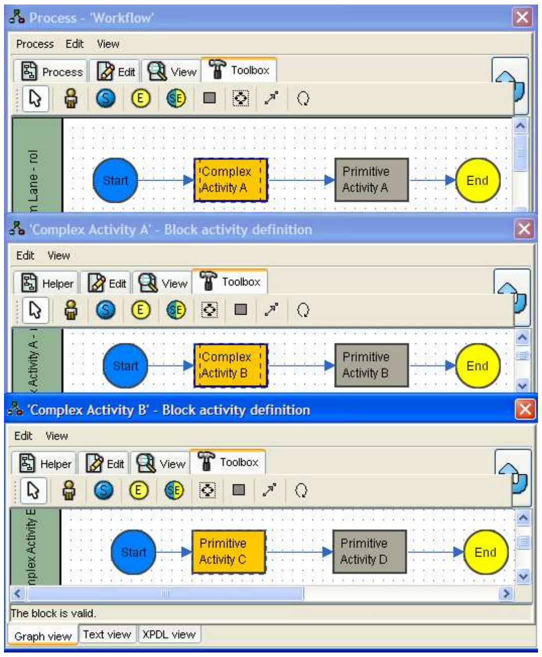


Fig. 2. Specification of Complex/Primitive activities in JaWE

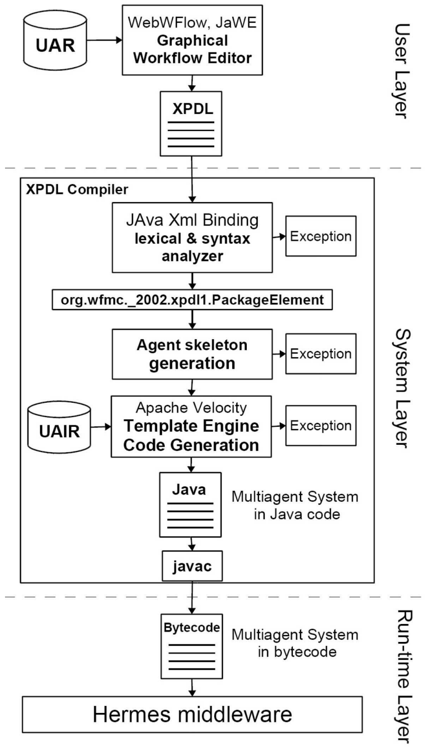

UAR
WebWFlow, JaWE Graphical Workflow Editor
User Layer
XPDL
XPDL Compiler
JAva Xml Binding lexical & syntax analyzer
Exception
org.wfmc._2002.xpdl1.PackageElement
Agent skeleton generation
Exception
UAIR
Apache Velocity Template Engine Code Generation
Exception
System Layer
Java
Multiagent System in Java code
javac
Bytecode
Multiagent System in bytecode
Hermes middleware
Run-time Layer

Fig. 3. Outline of workflow compilation process in Hermes

Furthermore, for security UserAgents can access a local resource only by interacting with ServiceAgent that is the "guard" of the resource. It follows a detailed description of the main components and functionalities of each layer.

A. User Layer

The user layer is based on workflow technology and provides to users a set of programs for interacting with the worklow management system. There are two main families of programs: programs for specifying, managing and reusing existing workflow specifications, and programs enabling administration and direct interaction with the workflow management system. The workflow editor is the program that supports the workflows specification by composing activities in a graphical environment. Hermes provides two editors, one is a plugin of the stand-alone JaWE [10] editor and the other is WebWFlow, a web-based editor. Both editors enable the specification of workflows by using XML Process Definition Language (XPDL) [14] a standard provided by the WfMC [12]. Activities used in a workflow are configured by specifying input parameters and their effects are recognizable as modification of state variables or modification on the environment's status. Workflow editors enable the composition of both primitive and complex activities. A primitive activity is an activity that can be directly executed. Users can specify primitive activity without knowing the real implementation. A complex activity is an activity that must be specified before it can be used; as Figure 2 shows the specification of a complex activity could be a workflow of complex and/or simple activities. By using complex activities the specification of workflows is simplified because they enhance both hierarchical specification and reuse: we can use an already existing complex activity without caring of its specification. Users can use complex activities and stored workflows to increase productivity when specifying new workflows. Moreover, large libraries of both domain specific primitives and complex activities can be loaded to specialize the editor for a specific application domain.

B. System Layer

System Layer, on the middle architecture, provides the needed environment to map a user-level workflow into a set of primitive activities. The execution of these latter is coordinated by suitable model, they implement the activities at user level and embed implementation details abstracted from the execution environment. These primitive activities are implemented by autonomous software entities UserAgent able to react to the environment changes where they are executed. A compiler generates a pool of user mobile agents from the workflow specification. Due to the lack of space, workflow compilation process shown in Figure 3 will not discussed here and we refer to [4] for further details.

C. Run-time Layer

Run-time Layer, at the bottom of the architecture, provides primitives and services essential for agent mobility and resources access. The kernel is the platform for mobile computing which provides primitives for discovery, mobility, communication, and security. As already described, the overall structure of the system is very complex, it supports abstract specifications that are mapped into a complex distributed and coordinated

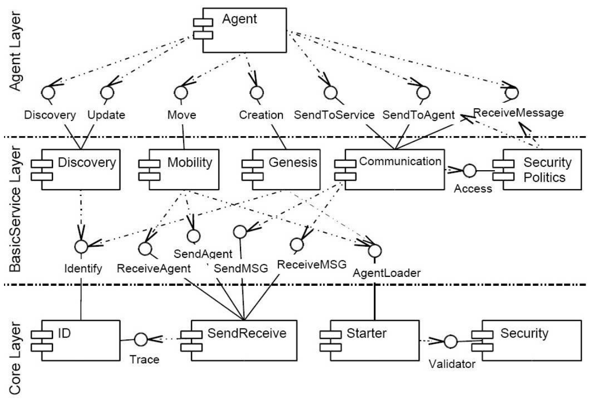


Fig. 4. 3-Layered Architecture of Hermes Mobile Computing Platform.

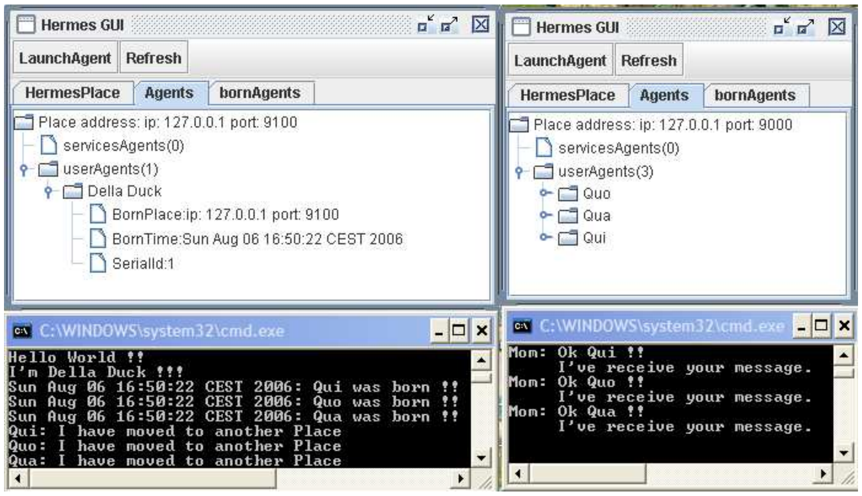
flows of activities over a large-scale distributed system. In order to master this complexity and to support the reusability of existing artefact during the development of a middleware system for a specific application domain, we designed Hermes kernel following a component-based [7] approach. Figure 4 shows the main components placed in the 3-Layered Architecture of Hermes Mobile Computing Platform. It follows a detailed description of components belonged to each layer.

1) Core Layer: It is the lowest layer of the architecture and contains base functions of the system, such as the implementation of the inter-platform communication protocols and agent management functions. This layer is composed of four components: *ID*, *SendReceive*, *Starter* and *Security*. The *ID* component, implements general identity management functions by managing a repository containing information about locally generated agents. This repository is accessed whenever we want to know the current position of an agent. The *ID* component is also responsible for the creation of the identifiers to be associated to new agents. These identifiers contain information about birthplace, date and time of the agent's creation. Agent localization is simplified by information contained directly in the *ID*, such as birth place. In fact, the birth place of an agent hosts information about agent's current location. A second important feature of the Core is the *SendReceive* component. This component implements low level inter-platform communication by sending and receiving messages and agents. By using traceability services offered by the *ID* component, *SendReceive* can easily update or retrieve the exact position of a specific user agent. The *Starter* component processes any request for agent creation. This particular component, in fact, take an inactive agent (just created or migrated), and checks it for the absence of malicious or manipulated code. These agents, before activation, are dynamically linked to all basic services of the platform. During execution the agent is isolated from the *Core* Layer by the *BasicService* layer. The *Security* component, as mentioned above, checks for the presence of malicious code or manipulations within agent code.

2) BasicService Layer: This layer has five main components: *Discovery*, *Mobility*, *Genesis*, *Communication* and *Security Politics*. The *Discovery* component searches and detects service agents. When a user agents wants to communicate with a service, it will ask the *Discovery* for the right identifier to use as the messages's receiver. The service detection strategy can be implemented in several ways; for example by a fixed taxonomy or by an UDDI [5], commonly used in WebServices application domain. The mobility component enables the movement of code across platforms [11], it implements the interface used by the *Agent* component and it accesses to components of the *Core* layer to send, receive and load agents. It is important to note that real communication between different locations can be achieved only through Core's *SendReceive* component, and then migration is independent of the type of used transport. Mobility consists on copy the agent i.e. its code and its current state and send it to the destination platform where it will re-started in a specific point (weak mobility). The local agent is destroyed. The *Communication* component makes possible to send and receive agent-directed messages both in an intra- and inter-platform context. Intra-platform messages are messages sent between agents and services residing in the same platform. Inter-platform messages are messages sent to agents residing in different platforms (our system does not allow for remote communication between user agents and service agents). The agent requesting the dispatch of a message does not need to know, effectively, where the target agent is; in fact, the *ID* is sufficient to post correctly a message. The *Communication* component uses one of the *Security Policy*'s interfaces to ascertain whether the specific *UserAgent* or *ServiceAgent* has the right privileges for communication. If an Agent is not authorized to use a service, the message is destroyed. Before accessing resources and services, an agent must authenticate itself. The identification is performed by sending a login message to a specific ServiceAgent, as consequence the *SecurityPolitics* component jointly with the *Communication* component intercept the message and unlock the communication. The *SecurityPolitics* component centralizes control of permissions, protects services and resources from the user agents, and provides the administrator with an easy way to manage all permissions.

The last component of the service layer is the *Genesis* component that enables agent creation. A special case of agent creation is cloning that is performed when it is necessary to create a copy of an existing agent. The two copies differ only for the agent identifier.

3) Agent Layer: The *Agent* Layer is the upper layer of the mobile platform, the *Agent* Layer, contains all service and user agents. This component has not any interface, but it has only several dependencies upon the *BasicService* Layer. The *Agent* component provides a general abstract *Agent* class. *UserAgent* and *UserAgent* classes extend this abstract class. ServiceAgent consists of agents enabling access to local resources such data and tools. User agents execute complex tasks and implement part of the logic of the application. Java programmers can also develop



```
Place address: ip: 127.0.0.1 port: 9100
  servicesAgents(0)
  userAgents(1)
    Della Duck
      BornPlace:ip: 127.0.0.1 port: 9100
      BornTime:Sun Aug 06 16:50:22 CEST 2006
      SerialId:1
```

```
Hello World !!
I'm Della Duck !!!
Sun Aug 06 16:50:22 CEST 2006: Qui was born !!
Sun Aug 06 16:50:22 CEST 2006: Quo was born !!
Sun Aug 06 16:50:22 CEST 2006: Qua was born !!
Qui: I have moved to another Place
Quo: I have moved to another Place
Qua: I have moved to another Place
```

```
Place address: ip: 127.0.0.1 port: 9000
  servicesAgents(0)
  userAgents(3)
    Quo
    Qua
    Qui
```

```
Mom: Ok Qui !!
     I've receive your message.
Mom: Ok Quo !!
     I've receive your message.
Mom: Ok Qua !!
     I've receive your message.
```

Fig. 5. Final result produced by McDuckAgent.java

UserAgents by using the API provided by Hermes Mobile Computing Library. Listing 1 shows a simple demo. A MkDuckAgent called *"Della Duck"* creates three sons *Qui*, *Quo* and *Qua* -lines 24 to 40- by cloning itself. After clonation each new agent start its behaviour calling *"afterCloning"* as initial method.

```
1   package samples;
2   import hermesV2.*;
3   import hermesV2.agent.*;
4
5   public class McDuckAgent extends UserAgent {
6
7     public McDuckAgent(String agentName) {
8       super("Della Duck");
9     }
10
11    public void init() {
12      reception(); //I enable the reception
13                   //of messages for the father
14
15      System.out.println("Hello World !!");
16      System.out.println("I'm Della Duck !!!");
17
18      Identificator temp=null,son1=null,
19                    son2=null,son3=null;
20
21      /*afterCloning is the first method
22        called after the clonation*/
23
24      try {
25        son1 = clone("afterCloning", "Qui");
26        System.out.println(new Date(
27                System.currentTimeMillis() +
28                ": Qui was born !!");
29        son2 = clone("afterCloning", "Quo");
30        System.out.println(new Date(
31                System.currentTimeMillis() +
32                ": Quo was born !!");
33        son3 = clone("afterCloning", "Qua");
34        System.out.println(new Date(
35                System.currentTimeMillis() +
36                ": Qua was born !!");
37      } catch (CloneException ce) {
38        System.out.println(ce);
39      }
40      Message m0=null,m1=null,m2=null,m3=null;
41      while (!(m1!=null && m2!=null &&
42             m3!=null)){
43        m0 = getMessageSynch();
44          temp = m0.getSenderAgentId();
45          if (son1.equals(temp)) m1 = m0;
46          if (son2.equals(temp)) m2 = m0;
47          if (son3.equals(temp)) m3 = m0;
48          System.out.println((String)m0.getObject());
49      }
50      /*The mother replies to sons*/
51      Identificator myId = getIdentificator();
52      m1 = new Message(myId, son1,
53                  "Mom: Ok Qui !! \n     "+
54                  "I've receive your message.");
55      m2 = new Message(myId, son2,
56                  "Mom: Ok Quo !! \n     "+
57                  "I've receive your message.");
58      m3 = new Message(myId, son3,
59                  "Mom: Ok Qua !! \n     "+
60                  "I've receive your message.");
61      try {
62        sendMessageToUserAgent(m1);
63        sendMessageToUserAgent(m2);
64        sendMessageToUserAgent(m3);
65      } catch (CommunicationException ce) {
66        System.out.println(ce.getMessage());
67      }
68    }
69
70    public void afterCloning() {
71      Identificator myId = getIdentificator();
72      PlaceAddress  myBPA = myId.getBornPlaceAddress();
73      int           myBPAPort = myBPA.getPort();
74      try {
75        int port = (myBPAPort == 9100) ? 9000 : 9100;
76
77        PlaceAddress  myMPA =
78                new PlaceAddress(myBPA.getIp(), port);
79        this.move(myMPA, "afterMoving");
80      } catch (MigrationException me) {
81        System.out.println("MigrationException" + me);
82      }
83    }
84
85    public void afterMoving() {
86      reception(); //I enable the reception
87                   //of messages for the son
88
89      Identificator myId = getIdentificator();
90      Identificator mother = getFatherIdentificator();
91      Message m = null;
92
93      try {
94          m = new Message(myId, mother,
95                  getAgentName() +
96                  ": I have moved to another Place");
97          sendMessageToUserAgent(m);
98      } catch (CommunicationException ce) {
99        System.out.println(ce.getMessage());
100     }
101     m = getMessageSynch(mother);
102     System.out.println((String)m.getObject());
103   }
104 }
```

Listing 1. McDuckAgent.java

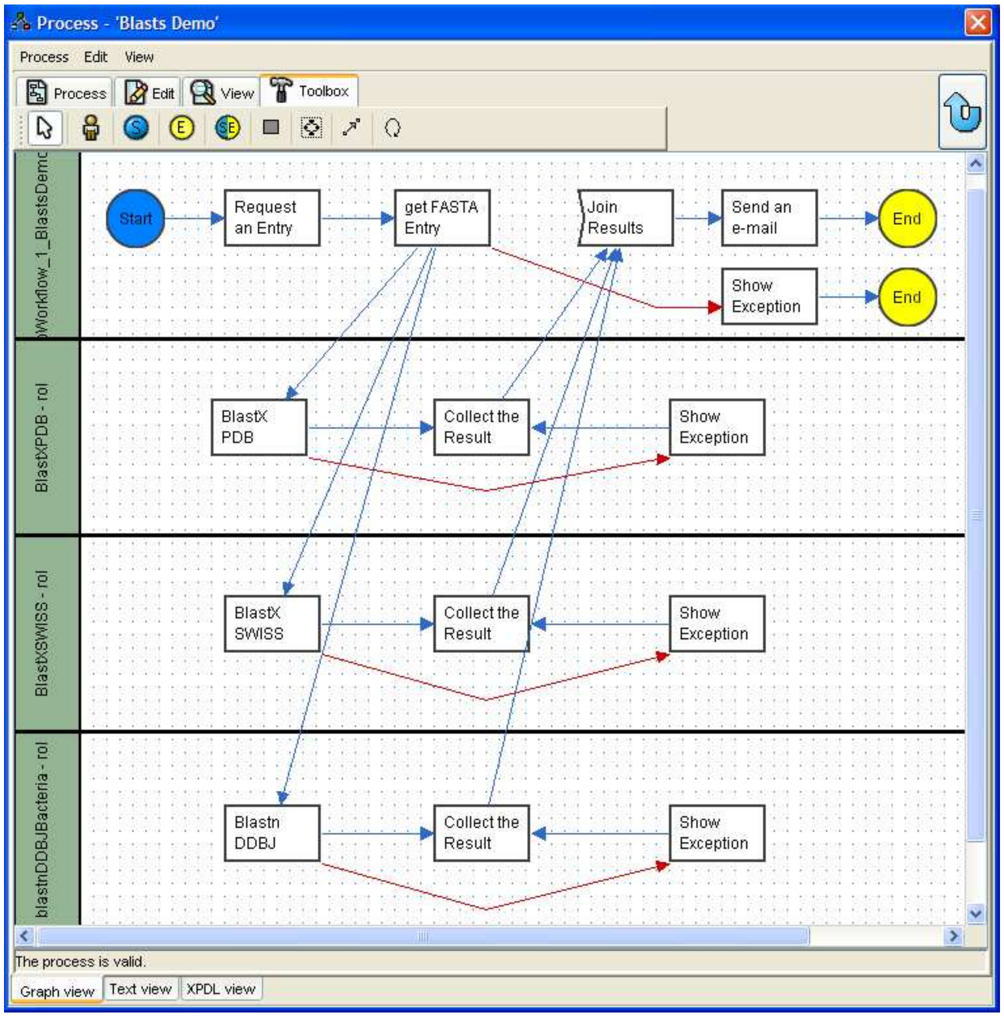


Fig. 6. Multiple blast workflow

By using *"move"* method -line 79- Qui, Quo amd Qua migrate to a *Place* different from where they were born. When they arrive in the new *Place* each one call the *"afterMoving"* -line 85- method. Then they notify to their mom their moving by using *"sendMessageToUserAgent"* -line 97- and *"getMessageSynch"* -line 101- methods. Figure 5 shows the final results.

D. Software requirements

One of the main features of Hermes middleware is its scalability. The present version, HermesV2, is a pure Java application whose kernel requires about 120KB of memory and interoperates across a systems ranging from microprocessors to very power workstations. The Hermes Mobile Computing Platform is available under LGPL on Sourgeforge [1] Web Site.

III. Model driven design and implementation of activity-based applications: a demo

In the present post-genomic era, biological information sources are crammed with information gathered from results of experiments performed in laboratories around the world, i.e., sequence alignments, hybridization data analysis or proteins interrelations. The amount of available information is constantly increasing, its wide distribution and the heterogeneity of the sources make difficult for bioscientists to manually collect and integrate information. In this section we present a demo of Hermes in the biological domain. In our example we want to find similar DNA sequences to a given one in several databases using Basic Local Alignment Search Tool [2] (BLAST). In particular, in this demo we want to compare, using BLAST, the nucleotide sequence in FASTA format of a given entry identificator with the sequences contained in the following databases:

- Protein Data Bank (PDB) [3]
- SWISS-PROT [4]
- DDBJ [5]

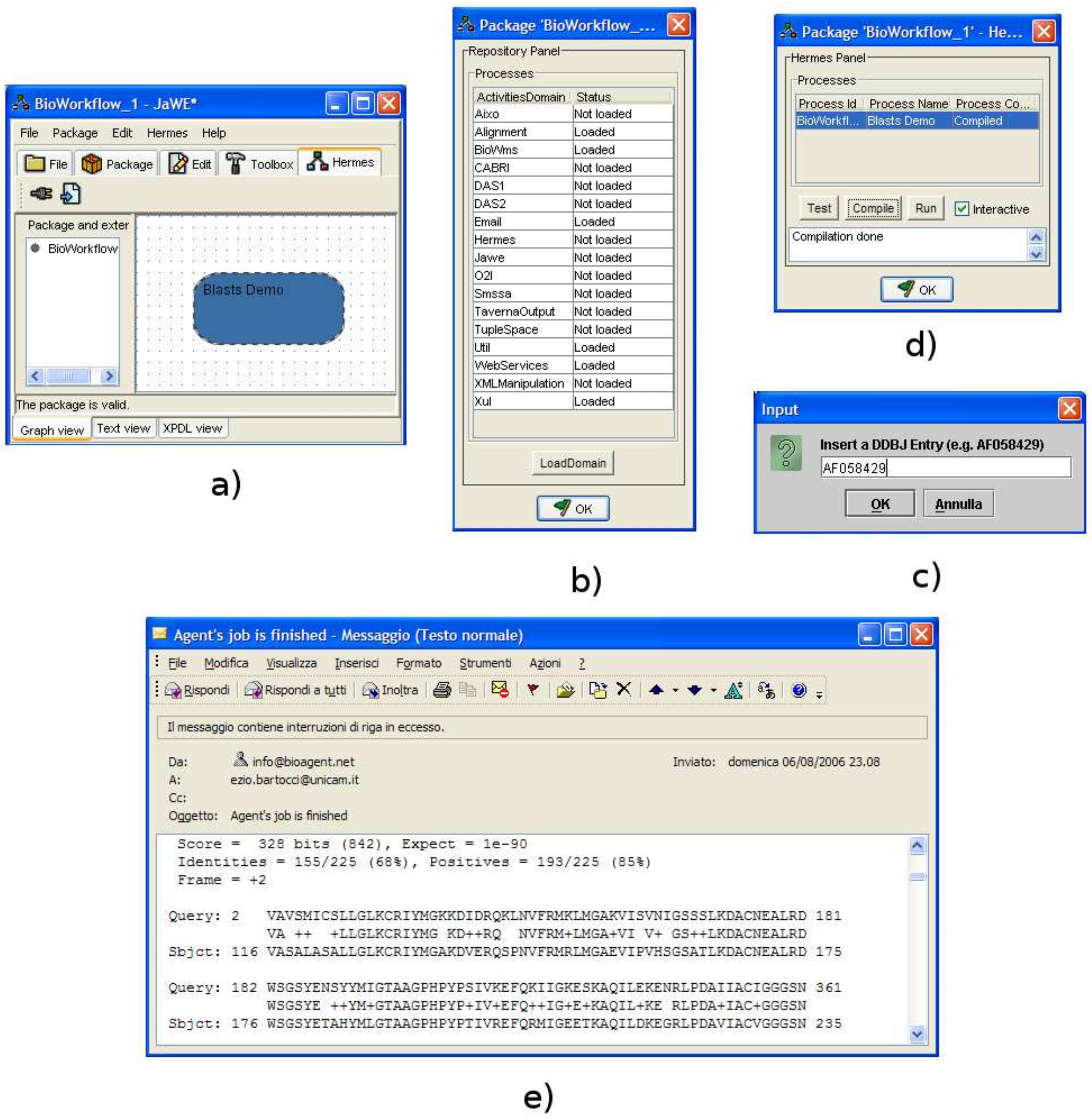
[1]http://sourceforge.net/projects/hermes-project

[2]http://www.ncbi.nlm.nih.gov/BLAST/

[3]http://www.rcsb.org/pdb/

[4]http://www.ebi.ac.uk/swissprot/

[5]http://www.ddbj.nig.ac.jp/


BioWorkflow_1 - JaWE*
File Package Edit Hermes Help
File Package Edit Toolbox Hermes
Package and exter
BioWorkflow
Blasts Demo
The package is valid.
Graph view Text view XPDL view
a)
Package 'BioWorkflow_...
Repository Panel
Processes
LoadDomain
OK
b)
Input
Insert a DDBJ Entry (e.g. AF058429)
AF058429
OK Annulla
c)
Package 'BioWorkflow_1' - He...
Hermes Panel
Processes
Test Compile Run Interactive
OK
d)
Agent's job is finished - Messaggio (Testo normale)
File Modifica Visualizza Inserisci Formato Strumenti Azioni ?
Rispondi Rispondi a tutti Inoltra
e)

ActivitiesDomain	Status
Aixo	Not loaded
Alignment	Loaded
BioWms	Loaded
CABRI	Not loaded
DAS1	Not loaded
DAS2	Not loaded
Email	Loaded
Hermes	Not loaded
Jawe	Not loaded
O2I	Not loaded
Smssa	Not loaded
TavernaOutput	Not loaded
TupleSpace	Not loaded
Util	Loaded
WebServices	Loaded
XMLManipulation	Not loaded
Xul	Loaded

Process Id	Process Name	Process Co...
BioWorkfl...	Blasts Demo	Compiled

Compilation done

Il messaggio contiene interruzioni di riga in eccesso.

Da: info@bioagent.net Inviato: domenica 06/08/2006 23.08
A: ezio.bartocci@unicam.it
Cc:
Oggetto: Agent's job is finished

```
 Score =  328 bits (842), Expect = 1e-90
 Identities = 155/225 (68%), Positives = 193/225 (85%)
 Frame = +2

Query: 2   VAVSMICSLLGLKCRIYMGKKDIDRQKLNVFRMKLMGAKVISVNIGSSSLKDACNEALRD 181
            VA ++  +LLGLKCRIYMG KD++RQ  NVFRM+LMGA+VI V+ GS++LKDACNEALRD
Sbjct: 116 VASALASALLGLKCRIYMGAKDVERQSPNVFRMRLMGAEVIPVHSGSATLKDACNEALRD 175

Query: 182 WSGSYENSYYMIGTAAGPHPYPSIVKEFQKIIGKESKAQILEKENRLPDAIIACIGGGSN 361
            WSGSYE ++YM+GTAAGPHPYP+IV+EFQ++IG+E+KAQIL+KE RLPDA+IAC+GGGSN
Sbjct: 176 WSGSYETAHYMLGTAAGPHPYPTIVREFQRMIGEETKAQILDKEGRLPDAVIACVGGGSN 235
```

Fig. 7. Multiple blast compilation and execution

The access to these databases is guaranteed by a set of Web Services. By using workflow editors a bioscientist can specify the logic order, as Figure 6 shows, of a set of domain-specific activities without knowing the related implementation details. Each rectangle is an activity and each swimlane represents a UserAgent. As Figure 7-b shows, user can exploit a set of previous defined domain-specific activities by importing the proper library.

BlastnDDBJ Agent, *BlastXSWISS* Agent and *BlastX-PDB* Agent receive the nucleotide sequence from the *BlastDemo* Agent and throught an interation with WSIF ServiceAgent, they compare the received sequence with sequences in each database using BLAST. If no exceptions occur, *BlastDemo* Agent join partial results and send the final document to user by email throught an interaction with the Email ServiceAgent. After saving this specification, you can reload -Figure 7-a-, compile -Figure 7-d- and execute -Figure 7-c and 7-e - the workflow previous defined.

IV. Some Case studies

The Hermes middleware has been validated in several projects. It follows a brief case study description of Hermes application in some of them.

A. *SI.CO.M project*

In the SI.CO.M [6] project we have developed a prototype based on Hermes middleware for the traceability of ichthyic products. Generally the product is traced throught the updating of databases distributed along the main sites of the weaving factory. This approach is not efficient because trace a faulty batch of products requires to query all databases, usually with an heterogeneous schema, of all sites interested in the production process. The proposed

[6]http://sicom.cs.unicam.it/

solution with the prototype named "TraceFish", exploiting the agent-based technology, allows to move automatically the information about a single batch from a site to another of the weaving factory overcoming the limits of the classical client/server approach.

B. O2I Project

The Oncology over Internet (O2I) [7] project is aimed to develop a framework to support searching, retrieving and filtering information from Internet for oncology research and clinics. Hermes in the context of O2I project is called Bioagent [8], it supports the the design and execution of user workflows involving access to literature, mutation and cell lines databases.

C. LITBIO Project

The main objective of the Laboratory of Interdisciplinary Technologies in Bioinformatics (LITBIO) [9] is to create infrastructure capable of supporting challenging international research and to develop new bioinformatics analysis strategies apply to biomedical and biotechnological data. To satisfy the most bioinformaticians needs we have proposed a multilayer architecture [2] based on Hermes middleware. At the user layer, it is intended to support in-silico experiments, resource discovery and biological systems simulation. The pivot of the architecture is a component called Resourceome [8], which keeps an alive index of resources in the bioinformatics domain using a specific ontology of resource information. A Workflow Management System, called BioWMS [3], provides a web-based interface to define in-silico experiments as workflows of complex and primitives activities. High level concepts concerning activities and data could be indexed in the Resourceome, that also dynamically supports workflow enactment, providing the related resources available at runtime. ORION [1], a multiagent system, is a proposed framework for modelling and engineering complex systems. The agent-oriented approach allows to describe the behavior of the individual components and the rules governing their interactions. The agents also provide, as middleware, the necessary flexibility to support data and distributed applications. A GRID infrastructure allows a transparent access to the high performance computing resources required, for example in the biological systems simulation.

Acknowledgment

This work is supported by the Investment Funds for Basic Research (MIUR-FIRB) project Laboratory of Interdisciplinary Technologies in Bioinformatics (LITBIO).

V. Conclusion

As the demo presented shows, Hermes middleware provides an integrated, flexible programming environment, whose user can easily configure for its application domain. Hermes is structured as a component-based, agent-oriented, 3-layered software architecture. It can configured for specific application domains by adding domain-specific component libraries. The user can specify, modify and execute his workflow in a very simple way. Workflow is specified abstractly in a graphical notation and mapped to a set of autonomous computational units (UserAgents) interacting through a communication medium. The mapping is achieved by compiler that is aware not only of contents of a library of implemented user activities but also the software and hardware environment to executing them. By using workflow as suitable technology to hide distribution and on mobile agents as flexible implementation strategy of workflow in a distributed environment, Hermes allows even to a not expert programmer a model driven design and implementation of a domain specific activity-based application.

[7]http://www.o2i.it/
[8]http://www.bioagent.net/
[9]http://www.litbio.org/

References

[1] M. Angeletti, A. Baldoncini, N. Cannata, F. Corradini, R. Culmone, C. Forcato, M. Mattioni, E. Merelli, and R. Piergallini. Orion: A spatial multi agent system framework for computational cellular dynamics of metabolic pathways. In *Proceedings of Bioinformatics ITalian Society (BITS) Meeting*, Bolgna, Italy, 2006.

[2] E. Bartocci, D. Cacciagrano, N. Cannata, F. Corradini, E. Merelli, and L. Milanesi. A GRID-based multilayer architecture for bioinformatics. In *Proceedings of NETTAB'06 Network Tools and Applications in Biology*, Santa Margherita di Pula, Cagliari, Italy, 2006.

[3] E. Bartocci, F. Corradini, and E. Merelli. BioWMS: A web based workflow management system for bioinformatics. In *Proceedings of Bioinformatics ITalian Society (BITS) Meeting*, Bologna, Italy, 2006.

[4] E. Bartocci, F. Corradini, and E. Merelli. Building a multiagent system from a user workflow specification. In *Proceedings of Workshop From Objects to Agents - WOA*, 2006.

[5] T. Bellwood, L. Clément, D. Ehnebuske, A. Hately, M. Hondo, Y. L. Husband, K. Januszewski, S. Lee, B. McKee, J. Munter, and C. von Riegen. UDDI version 3.0. Published specification, Oasis, 2002.

[6] D. Bonura, F. Corradini, E. Merelli, and G. Romiti. Farmas: a MAS for extended quality workflow. In *2nd IEEE International Workshop on Theory and Practice of Open Computational Systems*. IEEE Computer Society Press, 2004.

[7] D. Bonura, L. Mariani, and E. Merelli. Designing modular agent systems. In *Proceedings of NET.Object DAYS, Erfurt*, pages 245–263, September 2003.

[8] N. Cannata, E. Merelli, and R. B. Altman. Time to organize the bioinformatics resourceome. *PLoS Comput Biol.*, 1(7):e76, 2005.

[9] F. Corradini and E. Merelli. Hermes: agent-based middleware for mobile computing. In *Mobile Computing*, volume 3465, pages 234–270. LNCS, 2005.

[10] Enhydra. Jawe. http://jawe.enhydra.org/, 2003.

[11] A. Fuggetta, G. Picco, and G. Vigna. Understanding code mobility. *IEEE Transaction of Software Engineering*, 24(5):352–361, May 1998.

[12] D. Hollingsworth. The Workflow Reference Model, January 1995.

[13] E. Merelli, R. Culmone, and L. Mariani. Bioagent: a mobile agent system for bioscientists. In *NETTAB Workshop on Agents Nd Bioinformatics*, Bologna, July 2002.

[14] WfMC. Xml process definition language (xpdl). WfMC standard, W3C, October 2005.

The W4 Model and Infrastructure for Context-aware Browsing The World

Gabriella Castelli, Alberto Rosi Marco Mamei, Franco Zambonelli

Abstract—**The imminent mass deployment of pervasive computing technologies such as sensor networks and RFID tags, together with the increasing participation of the Web community in feeding geo-located information within tools such as Google Earth, will soon make available an incredible amount of information about the physical and social worlds and their processes. This opens up the possibility of exploiting all such information for the provisioning of pervasive context-aware services for "browsing the world", i.e., for facilitating users in gathering information about the world, interacting with it, and understanding it. However, for this to occur, proper models and infrastructures must be developed. In this paper we propose a simple model for the representation of contextual information, the design and implementation of a general infrastructure for browsing the world, as well as some exemplar services we have implemented over it.**

Index Terms—**Pervasive computing, Browsing the world, GIS, GPS, RFID tags.**

I. Introduction

Two apparently disjoint trends motivate this work. On the one hand, the imminent mass diffusion of pervasive computing technologies such as sensor networks [ChoK03] and RFID tags [Wan06] will soon make available an incredible amount of real-time information about the physical world, its processes, and its objects. On the other hand, the dramatic success of participatory Web tools (aka Web 2.0 technologies) is feeding the Web with information of any kind about any topic. In particular, mapping tools such as Google Earth and Google Maps get continuously enriched by geo-located information coming from very diverse social communities and related to a variety of facts and events situated in the world [But06].

Overall, both the above trends contribute to accumulate information that can be potentially used to build real-time and historical models of a number of facts and processes happening in the world. More pragmatically, the possibility of acquiring detailed digital information about the surrounding context opens up the possibility of exploiting all such information for "*browsing the world*" [Cas06]. The concept of browsing the world considers that, by properly integrating information about the surrounding world coming from both pervasive devices and form the Web, it will be possible for users to gather contextualized relevant information, and for services to effectively support user activities related to interacting with the physical world in a context-aware way.

However, considering that the amount of available information from a variety of sources could become overwhelming, its effective exploitation by users and services calls for proper models to represent such data in an expressive yet simple-to-be-manipulated way, and for proper software infrastructure to organize and provide access to it. Accordingly, the contribution of this paper is twofold.

First, we propose a simple model to represent contextual information about the physical world, for the use of both users' querying activities and context-aware services. The model, which we call "W4", is based on the consideration that most information about the world can be simply represented in terms of four "W"s – *Who, What, Where, When* – and that such a representation enables for very expressive, and flexible data usages.

Second, we describe the design and implementation of a general middleware infrastructure for browsing the world, facilitating the development and supporting the activities of general-purpose context-aware pervasive services. The infrastructure supports PDAs and laptops access to information coming from both pervasive devices and the Web, provides for representation and organization of data in W4 terms, makes available a Java interface for users' queries and for services access to such data, and it is integrated with both Google Earth and Google Map for the sake of effective user interfacing.

The remainder of this paper is organized as follows. Section 2 better details the general scenario of browsing the world and the challenges it implies. Section 3 presents the W4 model. Section 4 details the implemented software infrastructure. Section 5 presents some services we have implemented on top of our system. Section 6 discusses related work in the area. Section 7 concludes.

II. Browsing the World

In this section, we better define the scenario in which our research situates, by properly identifying the components involved in the "browsing the world" vision, and by discussing the associated key challenges.

A. Scenarios

As stated in the introduction, in the near future, our everyday environments will be densely populated by a variety

Manuscript received July 7, 2006. This work was supported in part by the project CASCADAS (IST-027807) funded by the FET Program of the European Commission.

Gabriella Castelli, Alberto Rosi, Marco Mamei, Franco Zambonelli are with the Dipartimento di Scienze e Metodi dell'Ingegneria. Università di Modena e Reggio Emilia, Italy; e-mail: name.surname@unimore.it.

of embedded devices such as sensor networks [ChoK03], RFID tags [Wan06]. Users in an environment will be able, via wireless interfaces mounted on some wearable computing device (e.g. a PDA or a smart phone), to directly access devices in their proximities to gather information about phenomena occurring in the surroundings or (as in the case of RFID tags attached to objects) about nearby physical objects. In addition, users will be able to access to the Web via some wireless communication technology, to dynamically retrieve any needed information. Other than accessing "traditional" Web information (e.g., html pages and Web services), this also enables users to access geo-located information concerning specific sites geographical areas and general facts and annotations about them, as they can continuously provided via collaborative Web 2.0 technologies by the Web community [Esp01, TerK06]. In addition, it enables users to access information generated by sensors and embedded devices (far in the world or close to him but beside his range of direct access).

Users, in turn, can decide to unveil (totally or to some limited extent) their presence in an environment, by making somehow available to the public their identity, location, and/or activities. This can occur by dynamically uploading such information on the Web, or by making it available to other via ad-hoc connections, or even by uploading it into surrounding pervasive devices. In this latter case, pervasive devices such as RFID tags would act as a sort distributed memory infrastructure [MamQZ06]. The location of users will be always available, either because they will carry on a GPS or because of location can be inferred by the patterns of access to pervasive devices (e.g., the access to a RFID tag with a known location implicitly determines the location of the user [Sat05]) (see Fig. 1).

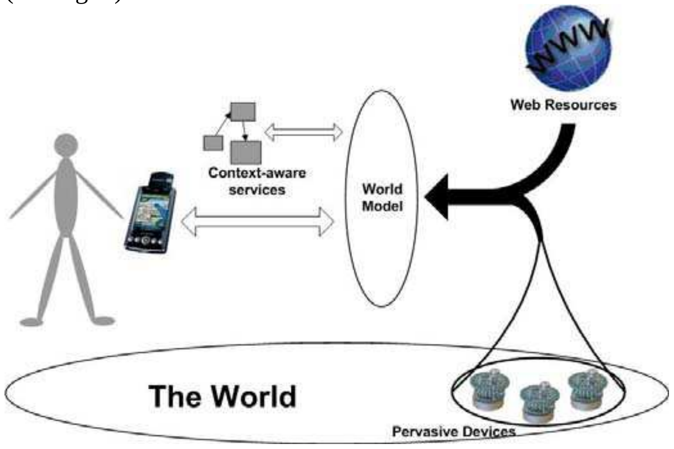


Figure 1. The general scenario of browsing the world.

On the basis of the above considerations, the concept of browsing the world, in general terms, consider the possibility of navigating in an information space that – by properly merging and integrating information coming from both pervasive devices and the Web can represent a detailed model of the world, comprising both present and historic fine-grained geo-located data about the world, its entities, its processes, and its social life. In context-aware user-centric terms, which are the ones of more interest here, the concept of browsing the world implies the possibility for users in an environment to access and navigate meaningful information about the surrounding physical world, and for software services to access and manipulate such information to enforce various degree of context-awareness and context-adaptation.

B. Challenges

From the merely technological viewpoint, the "browsing the world" vision could be already turned into reality. Indeed, novel services and Web sites that can be included in the "browsing the world" category appear every day [Cas06]. However, beside specific service implementations, for browsing the world to become common practice based on sound engineered activities, several challenges remains to be addressed. In particular:

1. It would be fundamental to create a general model to represent context information and to build a world model. Spatial information is important but it is not enough. Temporal information must be included, as well as information describing the activities taking place in the world. The model should enable to deal with incomplete information, and should allow navigation among context information on the basis of what is available at a given time.
2. It would be important to have a general infrastructure supporting the model that should work without requiring or committing to the availability of specific technologies. The infrastructure should be general-purpose, autonomic and adaptable. Relying on this infrastructure, the activities of browsing the world should not be compromised because of say, the temporal unavailability of an Internet connection or the unavailability of a GPS, or of an RFID reader. Consequently, applications built on that infrastructure should not mandate the availability of specific information, but should exploit whatever available information on a best effort basis.

Beyond the horizon, it would be important for such a general model to enable easy processing of data, to facilitate the identification of links between isolated bunch of information. This would enable the creation of complex knowledge networks, and possibly would promote the creation of "new knowledge", as it can be derived by inference from existing information [Bau06].

The attempt to face the above challenges, by defining a simple yet effective model for context data and a general software infrastructure, as preliminary and incomplete as it can be, is the exact goal of our work.

III. The W4 Context Model

We propose a simple model in which context data is expressed by a four field structure: who, what, where and when. Such a model appears effective in a number of circumstances since it points out some of the main topics that are also involved in human thinking: who is acting? What is he/she/it doing? Where and when the action takes place?

A. Overview

The goal of our proposal is to develop a general model to manage contextual information. Information to be handled will come from multiple and heterogeneous sources, and would be related to a large number of situations ranging from the description of physical properties in geographic areas, to social facts and processes happening in the world.

In particular, we developed a model in which context data is described by means of 4-fields tuple: (Who, What, Where, When). We chose this structure because of its evident meaning and flexibility. In fact, a W4-tuple allows to express a situation in a rather natural and human-like way, e.g., "someone or something (*Who*) does some activity (*What*) in a certain place (*Where*) at a specific time (*When*)". We call each of these tuples a *knowledge atom* to describe the fact they represent an atomic unit of context information.

Knowledge atoms are created by a number of software agents running on different (possibly embedded) devices, and will be stored in a suitable shared data space (in section 4, we detail our actual implementation of this space). Application agents that can range from context-aware service providers, to simple interfaces supporting users in browsing information, will access the shared space to retrieve those context information that are suitable for their application task.

B. W4 Data Representation and Generation

We define context as a four-field tuple (*Who*, *What*, *Where*, *When*):

- ***Who*** is the subject described by the context structure. *Who* may be a human person (e.g., Gabriella) or an unanimated part of the context (e.g., an RFID tag). The *Who* field is represented by a string with an associated namespace that defines the "kind" of entity that is represented. For example, valid entries for this field are: "person:Gabriella", "tag:tag#567".
- ***What*** is the activity performed by the subject. In the likely case that this is not directly available, it can be inferred from the other context parameters (e.g., an accelerometer can reveal that the user is jogging), or it ca be explicitly supplied by the user. This field is represented as a string containing a predicate-complement statement. For example, valid entries for the *What* field are: "read:book", "work:pervasive computing group", "read:temperature=23".
- ***Where*** is the location to which the context relates. In our model the location may be a physical space represented by coordinates (longitude, latitude) or by geographic regions (specifically, our model adopts the Postgis language to describe such regions [postgis.refractions.net]). Moreover, it can also be represented as a logical place. Logical places like "campus" or "bank" are mapped in the respective geographic by using a fixed dictionary. Logical places like "here" are mapped via simple algorithms considering the user current GPS location. Note that this enforces context-awareness, the same "here" information get multiple meanings depending on the user actual location.
- ***When*** is the time duration to which the context relates. It may be an exact range (e.g., "2006/07/19:09.00am - 2006/07/19:10.00am"), a concise description of a range (e.g., 9:28am), or even a logical value (e.g., "now", "today", "yesterday", "before"). Exact values are represented with a "begin-time-of-day – end-time-of-day" expression. Concise description and logical values are mapped via simple algorithms to the corresponding exact value. For example 9:28am = 2006/07/19:9:28am ± 5min. It is important to emphasize that concise time descriptions and logical times are contextual operators, their meaning depends on the time the query is actually issued.

Software agents are in charge of creating and inserting knowledge atoms in the shared space. Agents sense information from several devices (e.g. RFID tag, GPS devices, Web services) and combine them in order to produce a concise and effective description of what is happening in terms of a W4 tuple. The following examples illustrates the atom generation process.

Gabriella is walking in the campus' park. An agent running on her PDA can periodically create an atom describing her situation.

Who: user:Gabriella
What: works:pervasive computing group
Where: lonY, latX
When: now

The *Who* and *What* information are entered directly by the user at the login of the agent application, *Where* and *When* are dynamically provided by the GPS device.

Gabriella's PDA is connected with a RFID tag reader. A specific RFID agent controls the reader and handles the associated events. When a tag is read, the RFID agent creates a knowledge atom to store the tag information. In particular, either the tag would contain its own description, or the tag ID would be resolved in a dictionary to retrieve the description. This information, together with the "tag" namespace will fill the *Who* field. *What* is left unspecified. The agent accesses the GPS to retrieve location of the tag and fill the *Where* field. Finally, it completes the When field with the logical value "now".

Who: tag:statue of Ludovico Ariosto
What: -
Where: lonY, latX
When: now

C. W4 Interface

Knowledge atoms will be stored in a shared data space. In particular, our model relies on the following non-blocking and deterministic operations:

void inject(KnowledgeAtom a); enters a knowledge atom in the shared space
KnowledgeAtom[] read(KnowledgeAtom a); retrieves all the atoms matching a template knowledge atom.

The *inject* operation is trivial: an agent accesses the shared data space and store a knowledge atom there.

The *read* operation, instead, requires some more discussion. The W4 Model is suitable not only to represent context information, but for questioning too. A query will be represented by a W4 tuple with missing values (i.e., fields left unspecified). The read operation triggers a patter matching procedure between the query and the knowledge atoms that already populate the data space. Matching atoms are returned as results of the query. In this process, it is important to understand that the pattern matching operations work rather differently from the traditional tuple space model. In fact, our proposal can rely on the W4 structure to enforce more expressive pattern matching operations that have a different meaning for the various Ws.

- **Who and What.** Pattern matching operations in these two fields is based on string-based regular expressions. For example, a patter like "user:*" will match any user.
- **Where.** Pattern matching in this field involves spatial operations (again inspired by Postgis operations). Basically, the template defines a bounding box. Everything within the bounding box, matches the template. For example, a pattern like "circle,center(lonY,latX),radius:500m" defines a circle centered at (lonY, latX) with a 500m radius. Tuples with a *Where* field within the circle will match the template. Logical places have to be translated into actual spatial regions before of going through the pattern matching.
- **When.** In this kind of pattern matching, the template defines a time interval. Everything that happened within that interval matches the template. Concise time descriptions and logical times will be converted into actual time interval before of pattern matching.

The following two examples illustrate the querying process.
Gabriella is walking in the campus, and wants to know if some colleague is near. She will ask (read operation):

Who: user:*
What: works:pervasive computing group
Where: circle,center(lonY,latX),radius:500m
When: now

Analogously, Gabriella can ask if some of her colleagues has gone to work in the morning:

Who: user:*
What: works:pervasive computing group
Where: office
When: 2006/07/19:09.00am - 2006/07/19:10.00am

It is important to emphasize that returned answers have not to be "complete" W4 atoms. The pattern matching mechanism also allows matches between incomplete information. Thus, following this approach, applications are based on components entering complete and incomplete context information and getting in response other refined (but possibly still incomplete) information.

D. *Discussion and Future Extensions*

In our opinion the proposed W4 model addresses some of the previous challenges. First of all, the model is general enough to be used in different application fields, e.g. outdoor and indoor navigation, location-based services etc. The field structure is suitable fore a variety of application, indeed no field is application specific. The 4-field structure is supposed to have meaning for almost all context subject. Nevertheless, if a field is empty the others may carry useful piece of knowledge. The lack of a field doesn't compromise the correctness of an atom neither the ability of retrieving atoms. This makes the model autonomic and adaptable. Even if the structure is very simple, it conveys a lot of information beyond the spatial ones. The description includes both parameters of the context (time, location) and parameters about activities being undertaken.

In our opinion, there are however two important extensions that could be valuably added to the model.

On the one hand, the current model is somewhat limited by the lack of a reference ontology that could add semantic relationships to the concepts in the W-fields. With such an ontology in place, knowledge atoms could be related also if their fields do not match exactly, and also application agents would be able to manipulate the retrieved context information in a more meaningful way.

On the other hand, Although pattern matching operations proved rather flexible to retrieve context information, in our future work, we would like to exploit the W4 structure to better navigate the context repository. More specifically, we would like to link together the various knowledge atoms to form a knowledge network where it would be possible to navigate from one W4 tuple to the other. From this perspective, the W fields could be link to other knowledge atoms, so that it would be possible, for example, to follow the *Where* link to get further information on where a given entity is located. Our idea, is that the possibility of querying this network, instead of a flat tuple space, would allow much more semantically rich questions and inferences. In particular, new knowledge could be produced by navigating the knowledge network and combining and aggregating existing information into new knowledge atoms.

IV. The "Browsing the World" Infrastructure

To enable the concept of "browsing the world", we designed and implemented an infrastructure based on the W4 model. In this section we first present the general architecture underlying our infrastructure, then we will detail the parts that fulfill the W4 model.

A. *The W4 Architecture*

A general infrastructure to enable human-centric browsing of the world must include services for data acquisition, data integration, and data visualization. The architecture we have implemented is organized as follows:

1. Putting humans at the center, our architecture considers users with portable computing devices (i.e., laptops or PDAs), integrating localization devices (i.e., GPS), devices to acquire information from the physical world (i.e., RFID readers and sensors), and means to connect to the Internet (i.e., WiFi and/or UMTS connections).
2. Data coming from these devices (there included user GPS data) is represented by means of the W4 tuples, and stored in the local tuple space to be later accessed by application agents.
3. Relevant data are sent to a globally accessible shared tuple space containing the W4 model of the world. This space allows multiple users to exchange information and to conduct wide-area queries.
4. A RFID reader (in the form of a wearable glove) connected to the laptop or to the PDA via a serial cable

can be used to collect information from RFID tags dispersed in the environment. This information, enriched with the physical location where it has been collected (as provided by the GPS, device) is stored in the local tuple space.

5. Data coming from sensor network nodes (Crossbow MICAz) can be accessed by a suitable agent that collects sensed data and store them in the data space. Data is enriched with the physical location of the actual sensors and converted in the W4 format. Alternatively, sensor data could be collected by a base-station and sent directly to the "World" tuple space.
6. Specific services can be realized by means of application agents (i.e., autonomous software components) running locally on the user portable device and accessing, via the W4 model, both the local and "World" tuple spaces. Also, application agents can interface with a local GIS client (Google Earth or Google Maps) to turn data into a user-centric perspective.
7. Agents can dynamically connect to the Web to retrieve additional information to integrate with that coming from the W4 tuple spaces.

The whole system has been realized using the Java language. The "World" tuple space has been implemented through a Postgres database with spatial and temporal extensions. The local tuple space is simply implemented by a Java Vector. The RFID reader and the sensors are accessed via JNI and sockets respectively. User interface is provided by Google Earth (for laptops) and Google Maps accessed via the Minimo browser (for PDAs).

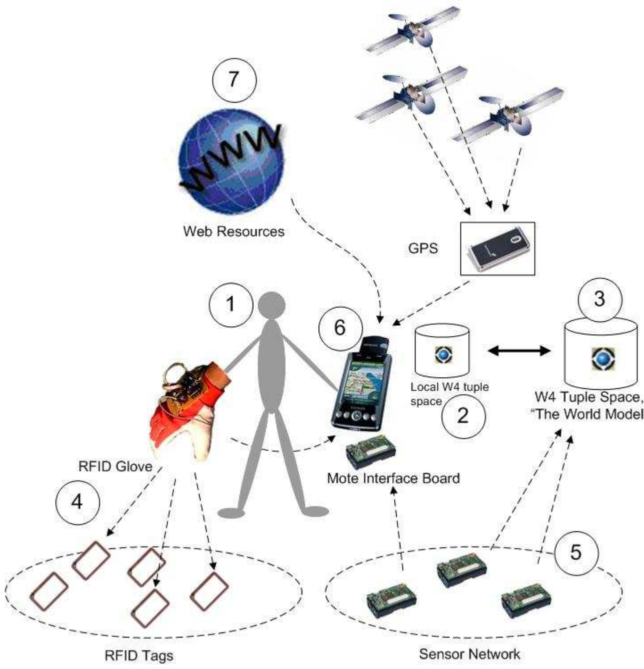


Figure 2. User centric infrastructure for browsing the world

B. W4 Tuple Space

All the information coming from the supported embedded devices (GPS, RFID and wireless sensors) is represented by means of W4 tuples, and stored in a local tuple space. Application agents access this space to retrieve W4 context information supporting their activities. Thus, application agents are completely decoupled from low-level embedded devices, and so they access and deal with contextual information only in tem of the W4 model. In addition, the availability of a local tuple space allows the system to work also in absence of a network connection and allows to minimize the generated data traffic (and its associated costs). Since this tuple space has to run on portable devices, it has been implemented by a simple Java Vector accessible with the W4 interface as described in 3.c.

Other than the local tuple space, our infrastructure is provided with a globally accessible shared tuple space containing a model of the world. This space allows multiple users to exchange information and to conduct wide-area queries. For scalability reasons, there could be more World tuple spaces, physically dislocated in the environment and linked in a DNS-like hierarchy. However, our current implementation consists of a single Web-accessible Tomcat server giving access to a Postgres database that store the W4 tuples. We realized JSP and Servlets implementing the W4 interface.

Application agents have to decide which information has to be sent to the World tuple space and which has to remain only locally confined. This decision may depend on many factor, such as privacy issues (e.g., a user may not be comfortable of constantly sending his GPS location on the Web) and scalability reasons. For example, trivial math says that storing one person entire life (100 years) GPS traces (2 floats) sampled at 0.1Hz amounts at 2.5 GB of highly redundant (thus compressible) data. Depending on needs, agents can decide the rate of data to send to the global server.

C. W4 Query Engine

The W4 query engine is the component that is in charge of managing the W4 queries and perform pattern matching operations.

The query engine running on the local tuple space has been developed in Java. It basically, scans the local Vector of tuples and uses String parsing methods and simple geometric algorithms (to handle *Where* clauses) for pattern matching.

The query engine running on the "World" tuple space dynamically translates W4 queries in SQL to execute them on the Postgres database. In this implementation, query pattern matching is supported either natively by SQL or by the Postgis spatial extension for the *Where* field.

It is worth emphasizing that the current implementation is only a first prototype and the current tuple space and query method is rather naïve. However, in future implementations, we will enrich the current infrastructure so as to manage, organize and integrate data in a more complex and clever way. In particular, as discussed in 3.4, we would like to abandon the current flat tuple-based implementation and structure context information in networks of knowledge. Such network-based representation would be more naturally distributable and could make our infrastructure more adaptive and autonomic.

D. The Graphical Interface

We developed a flexible graphical subsystem that can be easily employed on both laptops and PDAs. In particular, it interfaces with the GIS tools made available by Google: Google Earth and Google Maps to display retrieved context information as placemarks in a specific geographical area (see Fig. 3, 4, 5). Our graphical subsystem is based on the Keyhole Markup Language (KML), fully supported by Google Earth (at the moment only available for desktops and laptops), and at least partially supported by Google Maps and Google Maps for Mobile (that can be accessed also by PDAs and smart phones). This language allows to enrich geographical images coming from the Google GIS software with custom placemarks, images, 3D objects, etc. Thus, our graphical interface just translates proper W4 tuples in a corresponding KML file and dynamically provides it to the Google software. It is worth noticing that the KML language allows also to specify the user viewpoint on the map. This naturally supports context awareness, in that an agent could decide to center the map where relevant information are located.
Following this approach, application agents can then acquire relevant information by both interfacing with embedded devices, and relying on user interfaces. Such information will be represented in the W4 language for the sake of easy retrieval, manipulation and understanding. Finally, it will be converted in KML for the sake of effective visualization.

V. Application Examples

To test our model and infrastructure, we developed some simple applications highlighting the flexibility of the W4 model and infrastructure. In all these examples, we implemented a software agent that:

1. receives either static or dynamic queries from the user.
2. accesses the World tuple space to retrieve suitable context information.
3. creates a KML-formatted answer, and displays it either in Google Earth (for laptops) or in Google Map (for PDAs).

A. The Journey Map

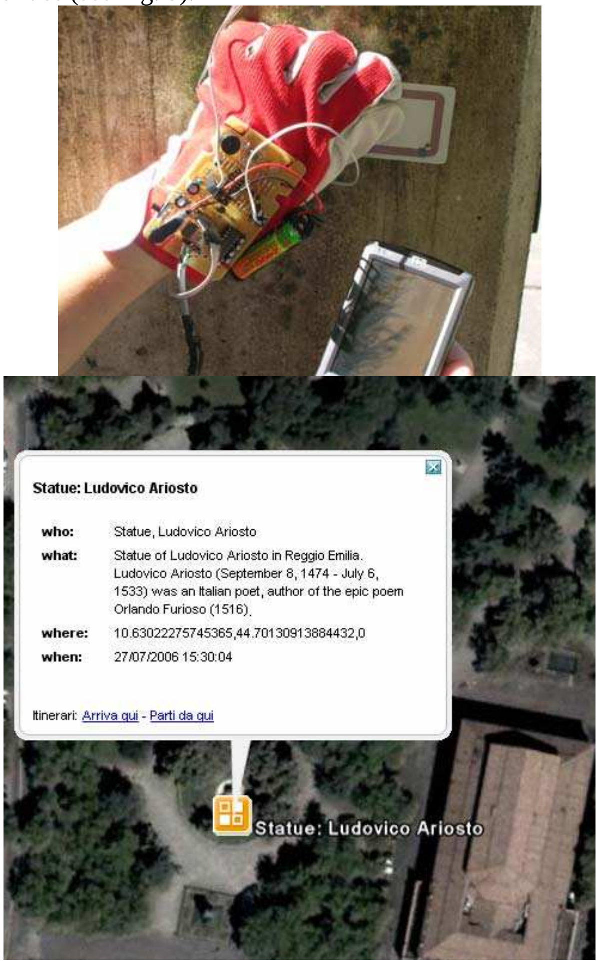
A first application allows to provide context-aware information to a user equipped with a GPS device and a RFID reader. In particular, we focused on the scenario in which a tourist wants to automatically build and maintain a diary of his journey. To this end, the proposed service allows to keep track of all the user movements and have them displayed on the map of the visited place. Moreover, the support for RFID allows to access likely-to-be-soon-available tourist information stored in RFID tags attached to monuments and art-pieces. From the diary perspective, this allows to store the visited art-pieces' location together with their description on the journey map. The W4 model can accommodate a number of interesting queries in this scenario. A first query allows to retrieve information about RFID tags being read.

Who: rfid:*
What: *
Where: lonY, latX
When: now

We implemented this as a static query that the agent asks cyclically to the local cache of tuple space (recall that the RFID agent is the one in charge of reading nearby tags and represent them in W4 format). If a tag is found, its content (properly parsed and enriched with Web-retrieved information) is used to create a KML placemark that will be displayed in the user interface (see Fig. 3). It is worth noticing that data coming from sensor network could be accessed via a similar W4 query.
Another service we realized for the journey map application allows an agent to recover user past locations from the World tuple space. This service could be useful to review a past tour and check the places where the user has been. The associated W4 query can be expressed in the form:

Who: user:Gabriella
What: *
Where: *
When: yesterday

Similarly as before, we implemented this service as a static query. The agent queries the World tuple space, retrieves a list of past GPS traces and displays as a KML-ployline in the user interface (see Fig. 5).



Statue: Ludovico Ariosto

who: Statue, Ludovico Ariosto
what: Statue of Ludovico Ariosto in Reggio Emilia. Ludovico Ariosto (September 8, 1474 - July 6, 1533) was an Italian poet, author of the epic poem Orlando Furioso (1516).
where: 10.63022275745365,44.70130913884432,0
when: 27/07/2006 15:30:04

Itinerari: Arriva qui - Parti da qui

Figure 3. (top) The RFID-reader embedded in a glove allows to identify tagged objects. (bottom) RFID tags becomes placemark with Web-retrieved information in the GIS software.

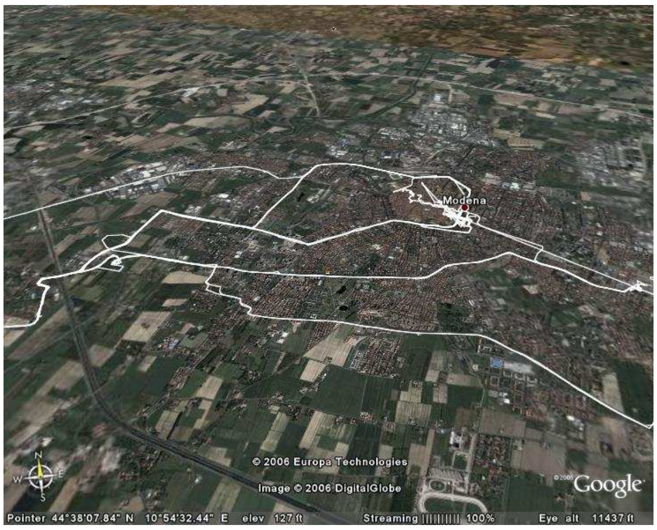

Modena
© 2006 Europa Technologies
Image © 2006 DigitalGlobe
Google
Pointer 44°38'07.84" N 10°54'32.44" E elev 127 ft
Streaming 100%
Eye alt 11437 ft

Figure 4. GUI showing user's past GPS traces

B. The People Map

A user equipped with a GPS device can decide to share his location with other users and, analogously, he may wish to be aware of the location of others users. For example, a group of friends can share their actual GPS locations (represented as knowledge atoms) with each other. This can happen either by uploading knowledge atoms to the World repository, or by exchanging them in ad-hoc way and storing them in the local tuple space cache only. Either way, collected knowledge atoms can be used to display users' locations on real-time a map (which, by the way, can highlight other interesting Web-retrieved information for the group, such as museums or bar, depending on the specific interests of the group). It is finally worth noticing that our current implementation of the service deal with privacy by leaving up to the individual user to decide whether to: share its position or not (and with which accuracy), make it available only to a restricted group of users, or to make it publicly available but only in an anonymous way.
The W4 model can accommodate the some relevant tasks with the following query

Who: user:*
What: works:pervasive computing group
Where: *
When: now

In addition, using the W4 model, we developed an advanced interface to enable location dependent queries. A user can use the "Who", "What", "Where" and "When" fields to dynamically compose queries and to ask information about local facts and "things" (e.g., "Find all restaurants within 500 meters") and get in answer the visualization at the correct location (i.e., in the form of Google Earth / Google Maps placemarks) of all that is found matching the query. Since the answer to a location-dependent query is based on the location of the mobile users, the results of these queries dynamically change as the users change their location in context-aware fashion. The query interface lets the user choose the number of unknown fields, potentially the user can access to the whole atom knowledge letting all the fields set to "any".

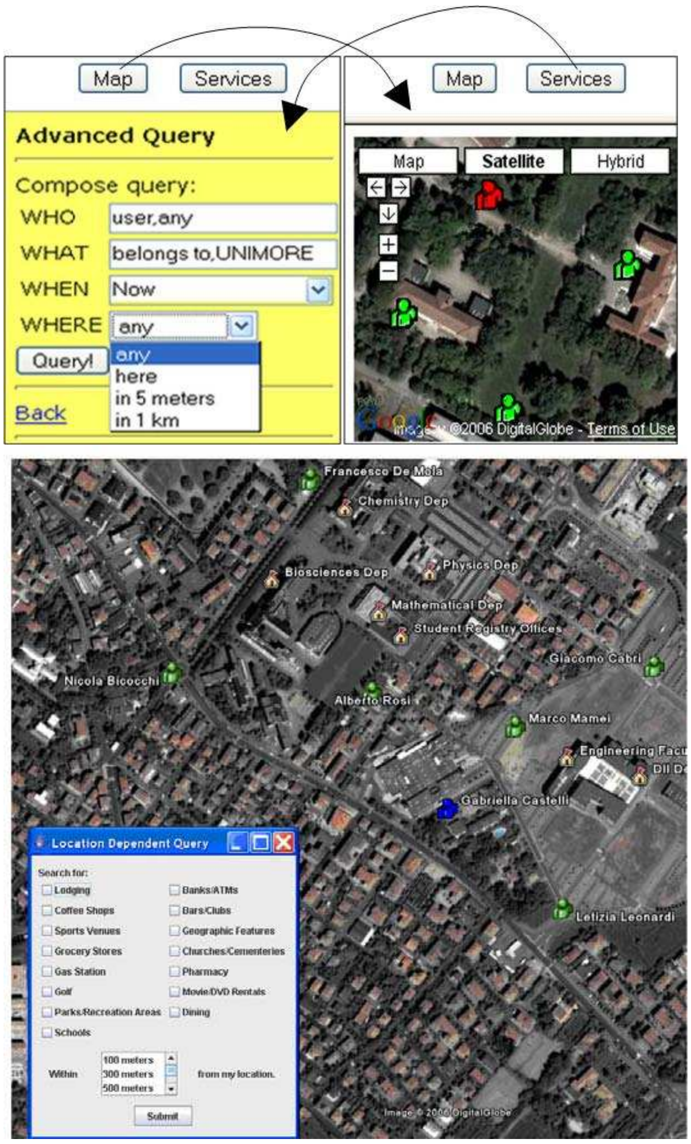


Figure 5. Map showing users real time locations with neighbor university facilities. (top) PDA user interface with the browser Minimo. (bottom) laptop user interface with Google Earth.

VI. Related Works

In the past few years, several models addressing contextual information and context-aware services have been investigated, and several infrastructure approaching -- to some extent – our concept of "browsing the world" have been proposed. In this section, we discuss and compare with ours some relevant proposals in these area.

A. Related Context Models

Existing researches on models for context-aware information try to create high-level and general-purpose context representation to be easily queried.
First works by Schmidt et al.[SchATT99] concentrate on the acquisition of context data from sensors and the processing of this raw data through a layered model. Similarly, the Context Toolkit [DeyAS99] focuses upon deriving context from raw

data by providing abstract component that can be connected together to capture and process the data from sensors. Although powerful, in our opinion, these approaches lacks of a common semantic to describe the data. This force developers to build new query languages depending on the kind of information at hand. On the contrary the W4 model provides a common semantic to deal with multiple context information in a coherent way.

We focuses upon develop a context model that can be easily queried. The pioneering work in this area is by Schilist et al. [SchAW94], who proposed a simple context model in which information are maintained by a set of environments variables. Analogously, Henricksen et al. in [HenIR02] analyze context adding the temporal aspect, information imperfection, various representation and high interrelation. However this approach leads to a long list of all characteristics of context, lacking in simplicity.

Others authors use structured context models as tuple space. In mobile computing several systems such as MARS[CabLZ00] and LIME[MurPR01] use the notion of reactive programming for shared unstructured tuple spaces. The Context Fabric' s fundamental abstraction is the InfoSpace [Hong02]. Each InfoSpace is a context tuple describing a single piece of context data in terms of entities (people, place, thing), attributes (e.g. the name) and relationship, special kinds of attributes that points to other entities. That approach doesn't address the temporal dimension of context. Egospaces [JulR02] provides a structured notion of context as name-value pairs in a Linda-like tuple space. Egospaces addresses context-aware programming in Ad-Hoc environments populated of agents by proposing an egocentric notion of context, i.e. every agent holds a personal representation of the world - that representation is called view. That approach is related to our approach with whom we share the idea of a structured representation of the context, however it is not concise because requires multiple tuples to represent a context piece. Indeed we don't have an egocentric notion of context.

Chang Xu et al. in [XuC05] propose a model very similar to our approach. It consists in a seven-field data structure that manage the description of the context. The fields are: subject, predicate, object, time, area, certainty, freshness, with similar meaning to W4. Beyond the field meaning, the purpose is different: their context model is not for browsing the world application. Similar considerations apply for the system described in [BraHCN06]: it describes RFID tags with the who-what-where-when structure. This approach is related to our with whom we share the idea of merging some information from different sources, e.g. the id from the tag with location from GPS or subject from the Web. However it's not a general model, since it is applied only to RFID tag.

B. Related Infrastructures

It is clear that a general infrastructures for browsing the world does not exist. Nevertheless, there exist several application-specific infrastructure and services which can show the importance of the problem. Some streams of works is simply based on representing Web information overlaid to geographical maps [But06, Rou05]. Examples include representations of avian–flu-outbreak reports [declanbutler.info/Flumaps1/avianflu.html], celebrity sightings [www.gawker.com/stalker], real estate information [www.forsalebyownercenter.com/google-earth-real-estate.aspx], and videogames [www.findskullisland.com]. Location-based services are natural candidates to use the power of novel GIS systems. A survey of novel tools to create location-based services is presented in [HarK05]. The presented systems combine GPS data, Web-search engines and GIS tools to retrieve and visualize services on a location basis. One system, for example, converts GPS data into street address by exploiting a standard geo-coding service (www.mapquest.com/features/main.adp?page=geocode). The address is then used to refine a search query submitted to a Web-search engine. Finally, the results are automatically displayed on a GIS tool. More dynamic applications, combine collaborative technologies (e.g., blogs and wiki) to GIS tools. MapWiki [TerK06], for example, is a Wiki collaborative environment where all the contents are located on a map. The contents can be edited and moved across the map and accessed on a location basis. Similarly, the Socialight software (socialight.com) allows a user to leave virtual Post-it notes, called sticky shadows, in specific sites around a city. The application checks the user actual coordinates with the note geospatial database and retrieves matching content.

In our opinion, all the above projects represents promising starting points of a future in which a wide range of information will be properly conveyed by novel GIS services. However, most of the above researches are special purpose and lack of a general architecture to manage and integrate pervasive, Web and GIS data. Furthermore, in opposition at our user-centric vision, their aim is to produce a centralized view of the world.

Other works concerned systems characterized by the presence of exploratory users and a surrounding environment. Users move forward the environment and access information exploiting different type of embedded sensors. Example of this systems are TinyLime [CuGG05] and all these system of world browsing like the system proposed in the past by our group [MamQZ06]. TinyLime is a middleware for wireless sensor networks that departs from the traditional setting where sensor data is collected by a central monitoring station, and enables instead multiple mobile monitoring stations to access the sensors in their proximity and share the collected data through wireless links. This context-aware setting is demanded by applications where the sensors are sparse and possibly isolated, and where on-site, location-dependent data collection is required. An extension of the LIME middleware for mobile ad hoc networks, TinyLime makes sensor data available through a tuple space interface, providing the illusion of shared memory between applications and sensors. At the same way our group describes the design and implementation of a tuple-based distributed memory realized with the use of RFID technology. The key idea is that everyday environments will be soon pervaded by RFID-tagged objects. By accessing in a wireless way the re-writable memory of such RFID tags according to a tuple-based access model, it is possible to enforce mobile and pervasive coordination and improve our interactions with the physical world. From a certain point of view we can consider these systems as “World Browsing” systems. Nevertheless we can say that our new system is certainly more complete and structured. These systems indeed base their functionality on specific technologies (in this case environmental sensors and RFID tags), they aren’t based on a well structured standard context model and further they don’t exploit information coming from the Web (as our does). A further interesting project is FLAME2008 [WeissVoG04]. Users through their PDA access to services (related to the 2008 Olympics games in Bijing) expressly fitted on their needs: FLAME2008 elaborates them on the base of activities and situations carried out by the user. The infrastructure is very interesting, in particular for its use of ontologies, and appears to be very complete. Nevertheless we noticed that it’s too bound to a specific application field and it doesn’t perform any mechanism for generate and store new knowledge (think at our mechanism of generating new knowledge atoms from performed queries in the past). Our model is certainly more user centric and location independent, besides it has been developed to adapt itself to a generic context, and above all, to be fully functional in any location with or without infrastructure support.

VII. Conclusions and Future Works

In the next few years, browsing the world will be as common as today is browsing the Web, and the increasing number of proposals and applications in this area definitely testify this trend. However, a number of challenging research issues still have to be faced to fully realize the vision. In this paper we present a simple model and infrastructure to handle some of these challenges. Our future research in this area will mainly focus on two aspects. On the one hand, we will try to integrate ontologies in our model to improve its expressiveness and flexibility. In particular, ontologies will allow more semantic forms of pattern matching. On the other hand, we will try to go further than the current flat knowledge atom representation and link knowledge atoms in suitable knowledge networks allowing a better and more semantic navigation of context information.

References

[BraHCN06] J. Bravo, R. Hervas, G. Chavira ,S. Nava, "Modeling Contexts by RFID-Sensor Fusion," percomw, pp. 30-34, Fourth Annual IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOMW'06), 2006.

[But06] D. Butler, “Virtual Globe: the Web-Wide World”, Nature, 439:776-778, Feb 2006.

[CabLZ00] Giacomo Cabri , Letizia Leonardi , Franco Zambonelli, MARS: A Programmable Coordination Architecture for Mobile Agents, IEEE Internet Computing, v.4 n.4, p.26-35, July 2000.

[Cas06] G. Castelli, A. Rosi, M. Mamei, F. Zambonelli, “Browsing the World: Bridging Pervasive Computing and the Web”, 2nd International Workshop on Ubiquitous Information Systems, Munster (D), September 2006.

[ChoK03] C.-Y. Chong, S. P. Kumar, “Sensor Networks: Evolution, opportunities, and challenges”, Proceedings of the IEEE, 91(8):1247-1256, Aug. 2003.

[CuGG05] C.Curino, M. Giani, M. Giorgetta, A. Giusti, A.L. Murphy, G.Picco: “Mobile Data Collection in Sensor Networks: The TinyLIME Middleware”; Dip. di Elettronica e Informazione, Politecnico di Milano, Italy and Dept. of Informatics, University of Lugano, Switzerland.

[CugP] G. Cugola, G. P. Picco, "PeerWare: Core Middleware Support for Peer-To-Peer and Mobile Systems", Technical report available at the URL: http://peerware.sourceforge.net/

[DeyAS99] Anind K. Dey, Gregory D. Abowd, Daniel Salber. A Context-Based Infrastructure for Smart Environments. Proceedings of the 1st International Workshop on Managing Interactions in Smart Environments, Dublin, Ireland, Dec 1999.

[Esp01] F. Espinoza, P. Persson, A. Sandin, H. Nystrom, E. Cacciatore, M. Bylund, "GeoNotes: Social and Navigational Aspects of Location-Based Information Systems", International Conference on Ubiquitous Computing, Atlanta (GE), 2001.

[HarK05] R. Hariharan, J. Krumm, E. Horvitz, "Web-Enhanced GPS", International Workshop on Location and Context Awareness, Munich (DE), 2005.

[HeyIR02] K. Henricksen , J. Indulska , A. Rakotonirainy, Modeling Context Information in Pervasive Computing Systems, Proceedings of the First International Conference on Pervasive Computing, p.167-180, August 26-28, 2002.

[Hong02] Hong. J. I., “The Context Fabric: An Infrastructure for Context-Aware Computing.”, Proc CHI 2002

[JulR02] C. Julien , G. Roman, Egocentric context-aware programming in ad hoc mobile environments, Proceedings of the 10th ACM SIGSOFT symposium on Foundations of software engineering, November 18-22, 2002, Charleston, South Carolina, USA.

[MamQZ06] M. Mamei, R. Quaglieri, F. Zambonelli, “Making Tuple Spaces Physical with RFID Tags”, ACM Symposium on Applied Computing, Dijon, FR, April 2006.

[MasCE01] C. Mascolo, L. Capra, W. Emmerich, “An XML based Middleware for Peer-to-Peer Computing”, 1st IEEE International Conference of Peer-to-Peer Computing, Linkoping (S), 2001.

[MurPR01] Murphy A. L., Picco G. P., and Roman G.-C., “Lime: A middleware for physical and logical mobility.” In Proc. of the 21st Int’l. Conf. on Distributed Computing Systems, pages 524–533, 2001.

[Rou05]W. Roush, “Killer Maps”, Technology Review, 11 September 2005

[Sat05] I. Satoh, “A Location Model for Pervasive Computing Environments”, 3rd International Conference on Pervasive Computing and Communications, IEEE CS Press, pp. 215-224, March 2005.

[SchATT99] A. Schmidt , K. A. Aidoo , A. Takaluoma , U. Tuomela , K. Van Laerhoven , W. Van de Velde, Advanced Interaction in Context, Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing, p.89-101, September 27-29, 1999, Karlsruhe, Germany.

[SchAW94] B. Schilit, N. Adams, and R. Want. Context-aware computing applications. In IEEE Workshop on Mobile Computing Systems and Applications, 1994.

[TerK06] Y. Teranishi, J. Kamahara, S. Shimojo, “MapWiki: A Ubiquitous Collaboration Environment on Shared Maps”, 6th International Symposium on Applications and the Internet Workshops, Phoenix (AZ), 2006.

[Wan06] R. Want, “An Introduction to RFID Technology”, IEEE Pervasive Computing, 5(1):25-33, 2006.

[WeissVoG04] N. Weißenberg, A. Voisard, Rüdiger Gartmann, “Using Ontologies in Personalized Mobile Applications”, GIS'04, November 12-13, 2004, Washington, DC, USA.

[XuC05] Chang Xu , S. C. Cheung, Inconsistency detection and resolution for context-aware middleware support, ACM SIGSOFT Software Engineering Notes, v.30 n.5, September 2005..

Mechanisms of Self-Organization in Pervasive Computing

Nicola Bicocchi, Marco Mamei, Franco Zambonelli

Abstract—**The mass deployment of sensors and pervasive computing systems expected in the next few years, will require novel approaches to program and gather information from such systems. Suitable approaches will be general purpose, independent of a specific scenario and sensor deployment, and able to adapt autonomically to different scales and to a number of unforeseen circumstances. This paper focuses on the requirements and issues of upcoming pervasive computing scenario, and surveys current research initiatives to deal with them. In particular researches addressing data retrieval and aggregation, macro-programming, and data integration in pervasive computing infrastructures will be detailed. Overall, the paper illustrates our ideas on collecting information from both sensor systems and Web resources and on linking them together in overlay knowledge network offering applications comprehensive and understandable information about their computational environment.**

Index Terms—**Pervasive computing, Sensor network, RFID tags, Self-organization.**

I. Introduction

In the near future, computer-based systems will be embedded in all our everyday objects and in our everyday environments. These systems will be typically communication enabled, and capable of coordinating with each other in the context of complex mobile distributed applications.

Current realizations of such scenarios, mainly in research labs, focus on special purpose systems, tailored for a specific application task. This specialization comes rather directly from the extremely limited capabilities of pervasive devices, that impose to rule out ancillary and general properties for the sake of optimization. In sensor network scenarios, for example, in order to be compliant with the thin battery budget of each sensor, applications rely on special purpose algorithms tailored for a specific sensors' deployment and for a specific set of data to be measured [WerL06].

Manuscript received July 7, 2006. This work was supported in part by the project CASCADAS (IST-027807) funded by the FET Program of the European Commission.

Nicola Bicocchi is with the Dipartimento di Scienze e Metodi dell'Ingegneria. Università di Modena e Reggio Emilia, Italy; e-mail: mamei.marco@unimore.it

Marco Mamei is with the Dipartimento di Scienze e Metodi dell'Ingegneria. Università di Modena e Reggio Emilia, Italy; e-mail: mamei.marco@unimore.it.

Franco Zambonelli is with the Dipartimento di Scienze e Metodi dell'Ingegneria. Università di Modena e Reggio Emilia, Italy; e-mail: franco.zambonelli@unimore.it.

In our opinion, such extreme specialization is transitory and more general-purpose approaches are likely to emerge soon. We think that future pervasive computing systems will be general purpose and users will be able to install and execute applications both on their private pervasive computing infrastructure (e.g., in smart home scenarios), and in publicly available ones (e.g., citywide infrastructures offering tourist information and services) [JonG05,Sri06]. In our opinion, this vision is motivated by the following considerations:

1. Advances in the manufacturing of pervasive computing devices (e.g., wireless sensors) will dramatically increase their performance, both in terms of computational capabilities and energy resources [Chu06].
2. Advances in energy-optimized and resource-optimized algorithms will provide efficient mechanisms to perform a number of basic services (e.g., routing), thus lowering the "resource-constraint-pressure" even further [Jon01].
3. Specialization hinders application development from a software engineering point of view. To create complex, dynamic and flexible services, it is mandatory to rely on general-purpose software infrastructures facilitating the programming task [Zam04].

All the above considerations show that general-purpose pervasive systems will be feasible in the next future, and will be required to offer advanced, flexible, robust and customizable services.

Given the extreme heterogeneity of future pervasive computing systems, their inherent dynamism and – most importantly – the incredible amount of data they will be able to produce, applications will have to autonomously adapt their behavior to different circumstances ranging from the scale of the pervasive network, to the quantity and granularity of information that will be available.

To achieve such a flexibility, applications will have to be highly context-aware (to understand and meaningfully interact with their environment) and, to this end, they will need to access properly represented contextual information.

In this direction, a number of recent researches try to represent contextual information by relying on overlay knowledge networks [Jel05, MamZ05, NagM04, Zam04]. Overlay knowledge networks can be regarded as distributed data structures encoding specific aspects of the application components' operational environment. Overlay knowledge networks are easily accessible by the components and provide easy-to-use context information (i.e., the overlays are specifically conceived to support their access and fruition).

The strength of these overlay knowledge networks is that they can be accessed piecewise as the application components visit different places of the distributed environment. This lets the components to access the right information at the right location.
From our perspective, "classic" overlay networks such as spanning tree and mesh data structures (i.e., routing distributed data structures providing components with a suitable application-specific view of the network) are particular examples of the more general concept of overlay knowledge networks [Jel05, IntG00, MadF02].
Overlay data structures such as fields and gradients [MamZ05], used in a number of macro-programming mechanisms [HadM06, NagM04], are another example of overlay knowledge network.
This paper is devoted to the above concepts and its main contribution is twofold:

1. We will better illustrate the scenario of general purpose pervasive computing showing its evolution and highlighting requirements and issues. In particular, we will discuss how considering the system as composed of a "continuum" of sensors and devices, rather than a discrete collection of them, may provide useful ideas and abstractions to deal with general purpose pervasive computing scenarios.
2. We will survey current research initiatives applying overlay knowledge networks to several autonomic and self-organizing pervasive computing applications. In particular, we will discuss how overlay knowledge networks could be suitable to the general scenario depicted above. By means of this survey, we will present how different research fields, ranging from data mining to distributed systems, are beginning to merge and complement each other to provide viable solutions to these novel scenarios.

The rest of this paper will be organized as follows. Section 2 details the upcoming scenario of pervasive computing and sensor networks, and illustrates the current shift from special-purpose and single-owner systems, to general-purpose and public pervasive infrastructures. Section 3 discusses issues and current approaches to program and gather information from pervasive distributed systems. In particular, it emphasizes the important role of overlay knowledge network in the majority of the proposals. Finally, Section 4 concludes the paper presenting some future research avenues in this area.

II. Scenario

As pointed out in the introduction, pervasive computing scenarios are moving toward general-purpose and widely available infrastructures that will enable a wide range of novel applications. In this section we are going to present the current setting of the scenario and its possible future evolution.

A. Current Setting

Recent advances in manufacturing and wireless communication are leading to the vision of pervasive and ubiquitous computing [But06, JonG05, Sri06]. The following technologies, currently widespread in research labs and likely to impact soon the real world, are the workhorses of this vision:

1. **Sensor networks** consist of several micro sensors scattered across an environment that collect environmental data (e.g. sound and temperature), process data (e.g., compute average and aggregate values) and wirelessly transmit such data to other sensors or base stations. The wireless sensor networks of the near future are envisioned to consist of hundreds to thousands of inexpensive wireless nodes, each with some computational power and sensing capability, operating in an unattended mode. They are intended for a broad range of environmental sensing applications from vehicle tracking to habitat monitoring. The hardware technologies for these networks (low cost processors, miniature sensing and radio modules) are available today, with further improvements in cost and capabilities expected within the next decade [WerL06].
2. **Radio Frequency Identification (RFID) tags** are small wireless radio transceivers that can be attached unobtrusively to objects as small as a watch or a toothbrush. Tags are extremely cheap and battery-free. Thus, they do not have power-exhaustion problems. Each tag is marked with a unique identifier and provided with a tiny memory allowing to store data. Suitable devices, called RFID readers access RFID tags by radio for read or write operations. The tags respond or store data accordingly using power scavenged from the signal coming from the RFID reader [Wan06, MamZ05]. For example, a mobile device detecting tagged objects nearby can build a sort of database of the objects available. This could have several applications in inventory and ware house management [LegT06].

In our opinion, these relatively static and hard-coded applications will be soon complemented by much more dynamic ones that will leverage sensors and RFID tags as a general, publicly-available infrastructure to "interface" with the physical world. Sensor data and RFID tags will be accessed by handheld devices we carry on everyday and will provide us with information such as crowded pubs nearby, dynamically-computed bus time tables and customized and useful information about objects and products around [MamQZ06, CurG05, Bor05, NatR06]. For example, RFID tags will possibly host scripts that will enable to tell how the data in it should be handled. This can enable forms of parasitic computing (the script is executed when a reader in range powers up the tag) [Rie06]. In addition, RFID tags can be coupled with sensors. A reader can power up the sensor that takes a measure and returns it to the reader [Wan04].

3. **Localization technologies** are key enablers for pervasive computing applications. Several mechanisms and technologies are currently proposed both for outdoor and indoor localization [HigB01, Sat05]. Location in the physical world remains the primary contextual information for almost all pervasive computing applications.
4. **The Web.** Given the ever improving coverage and

bandwidth of wireless network technologies, all kind of application scenarios could benefit from the ever increasing information available on the Web. For example, it is possible can find information about the small shop round the corner and discover the menu and the price list of that nice restaurant you have seen in that little village a few days ago. Still, the Web is missing connection with the physical world and with your actual physical location. So that a query as simple such as "where is the closest Chinese restaurant?" is something that current Web cannot answer satisfactorily. There is a lot of work in this kind of location-based services, but still some general purpose architecture to implement the idea is missing [Esp01, Eag05, HarK05].

On the basis of the above considerations, future pervasive infrastructures will be hosting several services and will integrate data from various sources, ranging from RFID, sensor networks and Web resources (see Fig. 1). Users in this scenario, will be able to access – via a number of handheld and wearable devices – several services dispersed in the environment.

- Users could query, either directly or via a proper base station, sensors in the environment to get various information such as traffic reports, weather conditions, and environmental parameters (e.g., temperature, light-condition) [Bal06, Sri06].
- Users could join profile matching services and applications. Profile matching applications consist of a sensor network composed of the smart-phones of the persons joining the application (note that a Bluetooth phone can be easily regarded as a wireless sensor, in that it can provide various data to other devices around). Such sensors will monitor their surrounding environment looking for nearby "compatible" persons and notify their users upon positive matches. [Eag05].
- Users could benefit of a number of automated pervasive services to complete economic transaction and acquire information. For example, RFID allows the vision of cashier-free retailers where a user just enters a retailer, takes what he needs and, when exiting, RFID readers installed at the retailer door read the items being taken and charge the customer credit card accordingly. RFID could also allow to store information where they will be most useful. For example, information on goods and products could be stored in RFID tags stuck at that product [Bor05, NatR06].
- Users could complement and integrate all the above data and information by means of suitable Web resources. For example, a sensor network detecting some kind of polluting agent could integrate collected data with a map showing nearby industrial implants to discover possible causes of the pollution, or in a map showing natural reserves to predict dangerous effects [JRDMS]. Similarly, a group of friends could decide to share with each other their actual GPS locations, and to display them on a map which highlights pubs and bars (coming from Web-based yellow pages) [Cas06].

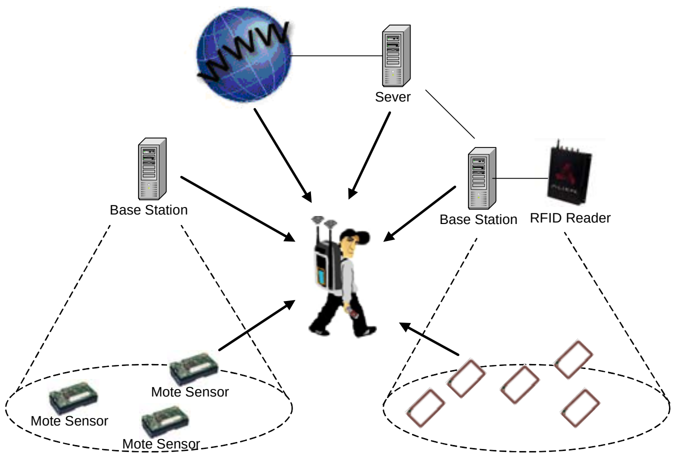


Figure 1. General pervasive architecture

B. Future Vision

The technologies described above could lead, in the next future, to a scenario in which sensors, actuators, memory and computational infrastructures will seamlessly wrap the real world. This will allow to collect and handle data coming from an unpredictable number of devices (sensors and Web resources) that will produce a sort of enriched perception of the world. With such an infrastructure in place, several interesting applications, in which users will be able to perceive the word beyond their five senses, will become feasible. For example, while walking on a street, it will be possible to perceive (i.e., get real-time information) on how much the restaurants nearby a crowded. In a similar way, it will be always possible to "sense" where friends and relatives are located, so as to arrange for meeting on the fly.
From our perspective, there are two main streams of research fueling this vision:

1. Novel approaches are needed to provide human users and application components with "extra-sensory" information without overloading their cognitive capabilities. With regard to human users, research on wearable computer is developing mechanisms to enable a person to see (by means of suitable see-through visors) computer-generated images overlaid to the physical world. Such images can augment the word by providing additional information [Dan06]. For example, they could show directions overlaid to the actual environment, or provide personal information overlaid to the person we are actually talking with. With regard to application components, suitable software infrastructures are needed to represent context information in a way that will be easy for the components to understand and use [MamZ05].
2. It is fundamental to actually store and manage that information at the infrastructure level. Research on RFID tags and sensors infrastructures, is a promising approach (complementary to the previous one) leading to this vision. In this context, the idea is to store and later retrieve information in the RFID tags and sensors that are likely to

populate (and saturate) our physical environment. Such an infrastructure could be used to enrich the world with context information that could be retrieved properly [MamQZ06]. For example, the infrastructure would allow to store "virtual" post-it notes across an environment to be found later on.

It is rather clear that such a vision implies a huge amount of information and data pervading the physical world that (given its scale) requires novel methodologies to be dealt with. In our opinion, a paradigm leading to the development of proper methodologies, in this context, could be based on the "continuum" abstraction [BeaB06]. Following this approach the system is designed having in mind a continuum of data sources (rather than a discrete network of devices) and so the abstraction being realized have to scale to an arbitrary number of devices. Of course to deal with such kind of large scale systems, autonomic and self-organization principles are needed [Dob07]. This is because managing the system at a fine-grained scale and addressing individual components will not be feasible (with the continuum abstraction in mind, the very concept of individual component tend to vanish), and so autonomic and self-organization mechanisms – where individual components manage themselves -- have to be introduced.

In particular, we envision an architecture, like the one depicted in Fig. 2. There, a countless number of sensors (wireless mote sensors, RFID, smart phones, and yet-to-come devices) enrich the world with digital information. This layer (represented as the bottom layer in Fig. 2) will be constituted by a huge number of heterogeneous and dynamically varying devices. The data at this basic level is at the finest possible granularity, and because of that will be hardly manageable and understandable by application components (i.e., too much data, too sparse knowledge).

Overlay knowledge networks are distributed data structures encoding specific aspects of the application components' operational environment. Overlay knowledge networks are easily accessible by the components and provide easy-to-use context information [MamZ05]. These overlay knowledge networks come into play to organize the data of the bottom layer into higher-level and more semantically expressive concepts. An example of this idea would be an overlay knowledge network that aggregates the data produced in a region of the underlying network to offer application components a single aggregated value (e.g., the average) representing the whole region. In other words, data produced by the bottom sensors can be aggregated at different level of abstractions. This aggregation produce discrete data elements each one managing portions of the continuum sensor space. These elements of the overlay knowledge network are represented in the higher layers of Fig. 2 and the upward arrows represent the process of creating higher-level concepts from low-level sensors.

This upward direction is not the only possible. In several situation, overlay knowledge network need to integrate and contextualize high-level concepts to a lower layer using sensor data. This integration is represented by the bottomward arrows in Fig. 2.

The resulting scenario is that of a hierarchy of an arbitrary number of overlays representing context information at different level of granularity. Application components, depending on their task, decide at which level to consider the context. Lower-level information will be aggregated to the proper level of abstraction. Higher-level information will be possibly contextualized to that level, and all this information will be integrated together in coherent view supporting application tasks.

Although the above description is at the level of modeling, and data aggregation, contextualization and integration mechanisms could be realized via whatever approach, in practice the model easily support a hierarchical architecture where higher-level servers collect and provide data at a certain level of granularity. Adopting this viewpoint, at the top level of Fig. 2, we have globally accessible Internet server providing worldwide aggregated information. At the lower layers, there are servers providing more and more specific data (e.g., state-wide, city-wide, building-wide data). At the bottom-layer there are the individual sensors offering extremely localized – but extremely detailed and up-to-date -- information.

Whatever the architecture, in order to realize the conceptual model in Fig. 2, it will be fundamental to rely on self-organization and autonomic principles. In fact, to guarantee robustness and scalability, the overlay knowledge network will have to maintain its coherency despite network glitches, sensors failures, the addition and removal of part of knowledge and other kind of contingencies.

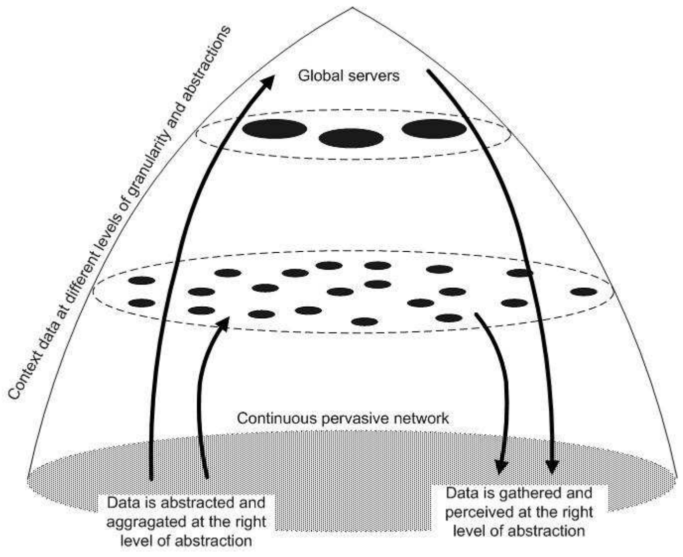

Context data at different levels of granularity and abstractions
Global servers
Continuous pervasive network
Data is abstracted and aggragated at the right level of abstraction
Data is gathered and perceived at the right level of abstraction

Figure 2. Continuum pervasive network with an arbitrary number of overlays describing context at different granularity

III. Issues and Current Approaches

Several new technologies and mechanisms are needed to fulfill the above vision and to create general purpose pervasive applications. In particular, we think that the main challenge is to provide applications with suitable overlay

knowledge networks to gather, understand and exploit context information at the proper level of abstraction for their application task. If a suitable context-representation is available, often the application task becomes easy, since application components see clearly from their context how to achieve the task [MamZ05].
From our perspective, there are three main research fields that are fruitfully tackling the above problems by exploiting overlay knowledge networks.

1. **Data Retrieval and Aggregation** comprises a number of researches trying to get data from a distributed sensors in an efficient way. In this context, overlay knowledge networks are used to create the routing structures to collect and aggregate data.
2. **Macro Programming** deals with programming a distributed system without explicitly defining single entities activities, but letting a compiler or a distributed middleware to translate high-level task into individual component activities in an automatic way. In this context, overlay knowledge networks are used to create regions and areas in a distributed systems allowing to suitably differentiate application execution disregarding individual components' activities.
3. **Data Integration** allows to integrate data from various sources (Web services and pervasive sensors) to offer application components an all-encompassing view of the operational environment (context). In this context, overlay knowledge networks are used to actually represent the integrated view that will be provided to application components.

In the next subsections we will present a survey of current research initiatives in these areas, showing also how the different areas themselves complements one another and pursue from different perspective the same ultimate goals.

A. Data Aggregation and Retrieval

The main goal of a sensor network (and of the majority of pervasive computing systems) is to collect data from the environment and to suitably present the data to application components. For this reason several researches try to devise mechanism to retrieve, collect and possibly aggregate data form a sensor network. The most common approach to collect data from the network consists in deploying data collector (i.e., sink) nodes which subscribe to some type of data flowing from sensing nodes about some particular phenomena. Once a data collector is registered to the network, each node starts to periodically send data to it. For example there may be a sink interested in receiving data from a particular region "A" between 2pm and 6pm if the temperature in that zone exceed 50°. Each day, during the selected time frame, sensors which detect temperatures over the selected threshold will send data to the sink. This is the simplest possible approach to retrieve data but has several disadvantages. In general since different sensor nodes detect the same phenomenon, it is likely that there will be an high degree of redundancy in the data flowing to the sink from different sources. Moreover each node located between a source and sink has to spend energy to route the message towards the destination. When compared to local processing of data, wireless transmission is extremely expensive. Researchers at the University of California, estimate that sending a single bit over radio is at least three orders of magnitude more expensive than executing a single instruction [ShrP04]. Last but not least, this approach is very sensitive to reading errors and sensors faults. If a node, broken or malicious, produces fake data, there is no straightforward way to filter it out.
To overcome the above problems, in-network filtering, processing and aggregation techniques can be used to conserve the scarce energy resources and improve data quality. From the information sink point of view in network data aggregation has two main advantages. The first one consist on a reduction of the potentially overwhelming data streams produced by the sensors. The second one, due to the activity of filtering and processing, is to reduce the complexity and the amount of data gathered letting further analysis more manageable. Probably, during the next few years, due to the increase of the size and density of sensor networks these advantages will quickly become determinant and every application will use some mechanisms where some sort of "in network" aggregation will be implemented natively.

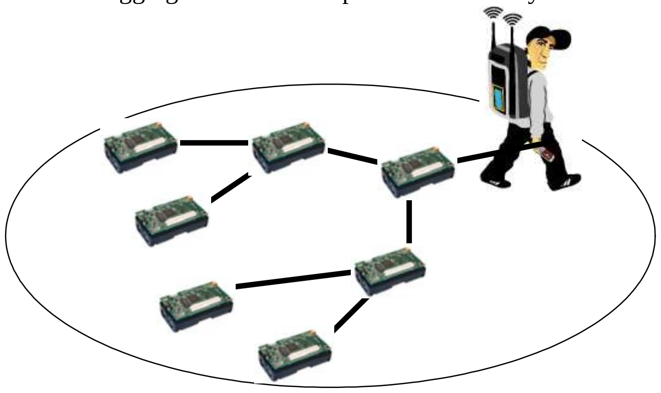


Figure 3. A spanning tree is created in the sensor network to route the collected data to a root node.

The work described in [Jel05] distinguishes *reactive* and *proactive* protocols for computing aggregate functions in a sensor network.

- *Reactive protocols* try to respond on demand to queries injected by nodes. If the answer is found in some region of the network, it is routed directly to the issuer node (see Fig. 3). Examples of this approach are well described in [IntG00, MadF02].
- *Proactive protocols* continuously provide aggregated data using some function and aim to diffuse meaningful values on every nodes in the networks in an adaptive way (see Fig. 4). "Adaptive" means that if sensed values change over time, the output of the algorithm should track variations reasonably quickly. Proactive protocols are often useful when aggregation is used as a building block for completely decentralized solutions to complex tasks [Jel05].

The above computation of aggregate functions is a key building block for many applications. In fact, aggregate data can be regarded as a simplified view of the components operational environment. Components may find simpler to access the aggregate value rather than distill the individual sensor readings.

Some examples of most used aggregated values are network size, average load, average uptime, location and description of hot spots, and so on. Local access to global information is often very useful, if not indispensable for building applications that are robust and adaptive. For example a fire alarm system has to trigger an alarm if the average temperature inside a building exceed a certain threshold or a distributed storage system has to know the overall free space over various device before processing a write() request. To reach the goal of a local access to global network features we have mainly two choices.

- The first one consists of gathering on some sinks all the (aggregated on not) sensor readings. After that we have to diffuse the global aggregated values into the overall network. This approach is simple and straightforward but has several serious limitations. The main one is the poor scalability. In fact as the network size grows, the amount of data that the sink has to manage become quickly overwhelming.
- On the other side we can use gossip based aggregations methods [Jel05]. Using this kind of algorithm local sensor readings are not to be convoyed to a sink, but can stay on sensors. The core of these protocols is a simple gossip-based communication scheme in which each node periodically selects some other random node to communicate with. During this communication the nodes update their local approximate values by performing some aggregation specific and strictly local computation based on their previous approximate values. After some iterations the local approximate value converge to the global value. The main advantages of these methods are that they are simple, scalable and provide local access to global values without any additional burden.

The last reported feature is really important in our vision. In a world full of sensors and actuators, users will need simple (i.e., aggregated) representations of the area of the network where they will be immersed. Using traditional routing based aggregation algorithm, due to their inherent “reactive” nature, will require, for each query, the building of a dedicated tree and to wait answers from an unknown number of sensors (which will may be very high). Instead, using gossip based algorithm, any user will be able to get, without any additional burden for the network, a simplified view of the area.

In general, the resulting aggregate value distributed across the network becomes an instance of overlay knowledge network. The overlay in fact extracts low level sensor reading to higher level concepts (i.e. aggregate values).

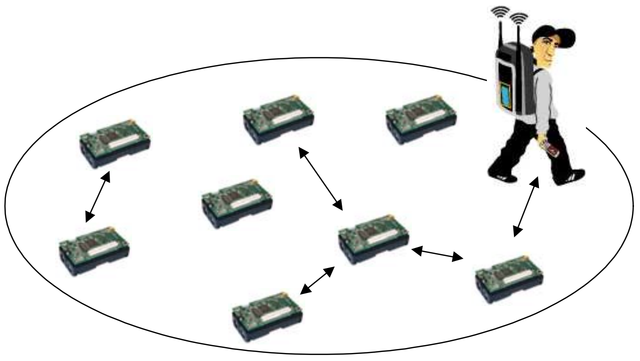


Figure 4. A gossip algorithms is run by nodes to aggregate data and report them back to an inquiring node.

In the next paragraph we briefly highlight some general examples of either reactive and proactive algorithm applications.

Data aggregation and retrieval is at the basis of a number of relevant application in the context of pervasive computing and sensor network. Currently the main application of sensor networks is environmental monitoring. This application consist of deploying a suitable number of ad hoc wireless connected sensors in a region. Such devices periodically read some environmental properties and route the acquired data towards a base station that is in charge of gathering and storing them. A good example of this kind application has been deployed on a natural reserve island in front of the Maine coast [Pol06]. There a hundred of sensors collect data from the birds nest, monitoring their micro climate. The data being collected are sent over the Internet and publicly available over the web.

Another promising application, which has not yet been fully developed, is object tracking. This activity consists of recognize and subsequently track moving targets over a monitored field. To achieve this task sensors do not have to collect massive amount of data to a central station for further analysis, but the network have to process sensed information and produce a simplified view of the physical world in which the object being tracked is readily visible. This application has been originally conceived in the military setting to drive vehicles in un-trusted areas. A promising new approach of this application involves multi sensory tracking. With this mechanism the same phenomenon can be recognized by means of different sensory inputs. For example, a car reaching a blind spot in a camera network could be tracked using sound sensors.

B. *Macro Programming*

A key challenge in pervasive computing is to provide powerful programming models to facilitate the development of applications in dynamic and heterogeneous environments.

One of the main conceptual difficulties is that we have direct control only on the agents’ local activities, while the application task is often expressed at the global scale [Zam04]. Bridging the gap between local and global activities is not easy, but it is possible: distributed algorithms for autonomous sensor networks like the ones presented in the previous subsection have been proposed and successfully verified, routing protocols is MANET (in which devices coordinate to let packets flow from sources to destinations) have already been widely used. The problem is still that the above successful approaches are ad-hoc to a specific application domain and it is very difficult to generalize them to other scenarios.

One promising research initiative in this direction is macro programming. The idea is to specify the global application tasks to be achieved and leaving to a compiler or a distributed middleware [HadM06, Nag02, NagM04] the tasks of mapping these global task into individual component activities. To build these languages there are two fundamental challenges:

- devise a global language suitable for a relevant class of applications
- devise a set of distributed algorithms to map the language into the component activities.

The above two tasks aim at hiding from the programmer low level details such as the heterogeneity and the scale of the underlying network.

In the last few years a number of research initiatives addressing macro programming have been proposed in several application scenarios.

In the Amorphous Computing project [Nag02], a macro-programming language is used to control shape formation in a reconfigurable sheet composed of thousands of identically-programmed, locally-interacting robotic agents. The desired global shape is specified at an "abstract" level as a folding construction on a continuous sheet of paper (i.e., origami). This construction is then automatically compiled to produce the program run by the identically-programmed agents. The global language allows to define the regions where the sheet has to fold, leaving to the compiler the identification of the low level action needed to actually reconfigure (i.e., bend) the robots.

Similar approaches for the control of shape and motion in a modular robot (i.e. a collection of simple autonomous actuator with few degrees of freedom connected with each other) have been recently proposed [StoN04, WerB06]. In these approaches a global description of the shape to be formed or of the gait to be followed is provided to the robot, either by representing the shape in some coordinate frame, or by adopting a description functionally specifying how the robot has to bend its actuators to move. Such a global description is then compiled into low level messages and actions to drive and coordinate the individual modules.

TinyDB [MadF02] and Cougar [YaoG02] provide a high-level SQL or XML-based query interface to sensor network data. The query is expressed by means of a high-level language indicating the data to be gathered in a declarative way. A compiler translates the query into the low-level sensor activities needed for the creation of the proper data collection and aggregation distributed algorithms.

Spatial Programming (SP) [Bor04] is a macro programming approach to program a sensor network. This approach allows to define regions in the network adopting a high-level semantic. In SP, for example, it is possible to address (and get a handle to) all the sensor in a given geographic region (described e.g. by its latitude and longitude). A low-level distributed middleware in then in charge to set-up suitable routing structures to actually address the proper sensors.

Abstract Regions (AR) [NewA04] is another macro programming approach to define regions in a sensor network. Rather than focusing on geographic regions like in Spatial Programming, AR focus on network regions (e.g., x-hop neighbors, spanning tree and planar meshes). A high-level language allows to specify the network region, while low level algorithm create the actual routing structure to handle the proper nodes.

Regiment is a functional macro programming [WelN04] language that generalize both the previous approaches. Regiment allows to define regions in the network able to represent spatially distributed, time-varying collections of node state. The programmer uses the language to express interest in a group of nodes with some geographic, logical, or topological relationship, such as all nodes within k radio hops of some anchor node. A distributed middleware is then in charge to map the regions into suitable sensor-level coordination protocols. Similar approaches to define regions in a distributed system according to spatial and functional characteristics have been presented in [BecH04]

A more comprehensive survey of currently proposed macro-programming languages can be found in [HadM06].

In general, all the reported macro-programming approaches uses suitable overlay knowledge networks to control the distributed program. In most of the proposals, overlay knowledge networks are used to define the regions where the components activities will be different. In Spatial Programming, for example, the overlay knowledge network is represented by the data structure identifying the region where data should be collected by the application.

To create complex, dynamic and flexible services, it is mandatory to rely on general-purpose software infrastructure facilitating the programming task. The ability to program a distributed system without explicitly and directly defining individual entities' activities will be a fundamental asset in this direction.

C. Data Integration

Pervasive computing applications will be naturally integrated with Web services and Internet resources. Not only Web services will be a natural technology to access pervasive applications remotely, but it could also provide further context information to the pervasive device. For example, sensors could get from the Internet the average temperature of the region they are in, and compare their sensor readings with that average. With this regard, we think that in the next future application will integrate together data coming from the Internet and data coming from the real world (sensors) and actually merge it together in a coherent framework providing advanced context-aware applications.

In this context, overlay knowledge networks are used to merge the collected data together, and to provide such data to application components in a coherent view.

A number of recent projects from different research communities (data mining, distributed systems, semantic Web,

Web services, etc.) are tackling the challenge of data integration across multiple providers.

One interesting research in this area is described in [PerP04]. The goal of this project is to develop a context-awareness system to detect and infer domestic activities performed by the users. The proposed approach is to infer the activities of the user on the basis of the objects he touches. For example, by sensing that the user touches a "teapot", some "teabags", "glasses" and "spoons", the system can infer that the user's action is "making tea". This kind of knowledge could be of use in a number of smart-home scenarios. To implement such an idea, the system relies on RFID tags associated to (and identifying) everyday objects, and gloves integrated with RFID reader worn by the user. This allows the system to detect, rather naturally, what the user is touching.

This stream of data coming from pervasive devices requires models of activities to detect what the user is doing. Such models are automatically mined from the Web. In particular, the system connects to specific "How to" sites, describing how to perform a specific activity, extracts the labels associated to the object being used, and creates a Bayesian network describing probabilistically the objects involvement in the different activities. The model is finally, checked against the data coming form the RFID reader to infer the activities being carried on.

In our opinion, this project is a perfect example of the fact that pervasive and Web resources complement each other, and by integrating them, it is possible to obtain novel and powerful services.

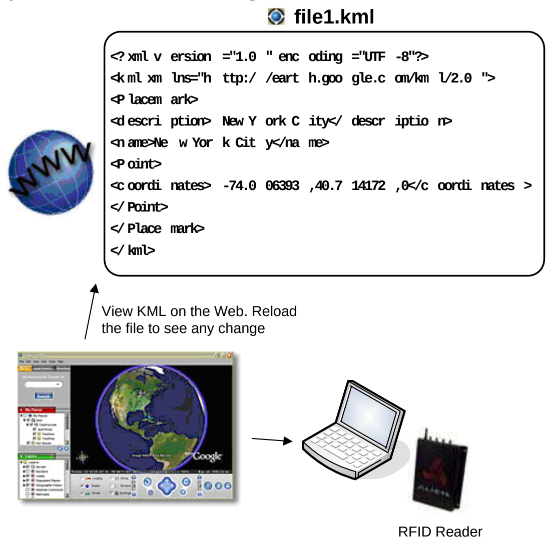
Another relevant approach is presented in [Eag03]. The goal of this work is to infer users context by capturing their speech. The voice of the user is record by a PDA carried on by the user. The voice signal is sent over a wireless network to a server that process the signal and transcribes the speech. The server connects to a Web service called Concept Net [Liu04] that is based on a knowledge network describing common-sense activities. Concept Net is, in fact, a huge repository of commonsense sentences (e.g., you'd order food in a restaurant) and a suitable API to access and mine the repository.

By providing ConceptNet with the speech transcription, the service is able to infer the most likely context for the user. For example, the speech: "Hi, today I'm going to have a cheeseburger and a beer" would let ConceptNet infer that the user context is "ordering food at a resturant". Such information is then sent back to the PDA for further actions.

Another interesting mechanism to combine sensor data and Web information involves the usage of GPS as sensors and Web-retrieved maps from open GIS-tool like Google Earth (http://earth.google.com). In [Cas06], we describe two services in this direction. A first service allows a user equipped with a RFID reader and a GPS device to see his actual location and past movements, and to dynamically create Google Earth placemarks of the tagged objects being read with the RFID reader at the right location. This service can be fruitfully employed in a number of situations. In particular, we focused on the scenario in which a tourist wants to automatically build and maintain a diary of his journey. To this end, the proposed service allows to keep track of all the user movements and have them displayed on the map of the visited place. Moreover, the support for RFID allows to access likely-to-be-soon-available tourist information stored in RFID tags attached to art-pieces. From the diary perspective, this allows to store the visited art-pieces' location together with their description on the journey map. In addition, our service could also provide with important logistic information. For example, the action of reading the tag of the user's car at a certain location triggers a new car-placemark on Google Earth showing the actual position of the car. This allows the tourist to easily recall where the car has been parked.

Another service, allows multiple users to share their list of placemarks and their current location. Again, this service can be employed in several scenarios, and we focused on supporting a group of tourists cooperatively visiting a place. Such a situation applies to a class of students or to a group of boy-scouts, where each person can visit the place independently, while keeping in touch and sharing information with the other members. To this end the service allows to share GPS data with other members and with the group leader (e.g., the teacher may be in need of monitoring the location of all the students). Moreover, placemarks pointed by one person may be shared across al the group. This can be useful to share opinions or interesting sightings, but also to easily agree on some meeting points. For example, by sharing placemarks, all the users can spot a suitable place (e.g., a pub) that is in the middle of them and agree to meet there (see Figure 5).

Other approaches in this direction, developed by other research groups, [PatL04] combine GPS data and maps to create a probabilistic model of the user activities. This approach allows to the system to learn the user motion routine (e.g., where does he go, where does he park the car, etc.) and possibly to check anomalies against the learned trend.


file1.kml

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.0">
<Placemark>
<description>New York City</description>
<name>New York City</name>
<Point>
<coordinates>-74.006393,40.714172,0</coordinates>
</Point>
</Placemark>
</kml>
```

Figure 5. Integration of GPS data and Web maps.

Finally, another source of information that researchers are trying to integrate is that coming from images widely available and tagged by services like Flickr (www.flickr.com). The idea at the core of some recent researches is to try to match pictures taken from cameras with those available on the

Internet. This would allow to get information about objects without the need of tagging them artificially. For example, the image of a tower taken by a camera phone could be matched against a data base of images to properly recognize it as the Pisa leaning tower [Jia06].

All the above examples show rather clearly that the approach of integrating resources and data from pervasive systems and Web resources in a promising research avenue.

IV. Conclusions

In this paper we presented our vision for next future pervasive computing systems. In our opinion, these systems will be general purpose and users will be able to install and execute applications both on their private pervasive computing infrastructure (e.g., in smart home scenarios), and in publicly available ones (e.g., citywide infrastructures offering tourist information and services). Given the extreme heterogeneity of this scenario, its inherent dynamism and – most importantly – the incredible amount of data the system will be able to produce, applications will be required to match and comply those characteristics. Applications will have to autonomously adapt their behavior to different circumstances ranging from the scale of the pervasive network, to the privacy-level being requested by the users. To achieve such a flexibility applications will have to be highly context-aware (to meaningfully interact with their environment) and autonomic. To this end, they will be able to gather relevant context information both from the pervasive network sensing the environment and from global-accessible Internet services. We also introduced how considering the system as composed of a "continuum" of sensors and devices, rather than a discrete collection of them, may provide useful ideas and abstractions to deal with the above challenges.

In addition, we presented the key mechanisms and researches trying to fulfill the above vision:

- Retrieve and aggregate data will provide developers with advanced tools to get data from a distributed system in an efficient way.
- Macro Programming a distributed system deals with programming a distributed system without explicitly defining single entities activities, but letting a compiler or distributed middleware to translate high-level task into individual component activities. This will allow developers to design systems composed of a huge number of components that will be able to carry on complex coordinated activities.
- Integrate data gathered from various sources allows to offer application components a coherent view of their context.

In particular, we tried to present how the concept of overlay knowledge networks may be at the basis of most of the proposal, and how overlay knowledge network may represent a framework to develop applications in future pervasive computing scenarios.

In our opinion, these researches are only at the beginning of addressing satisfactorily the requirements of future scenarios and several questions remain open: How to represent context information in a general way? How can we retrieve and access such huge amount of knowledge? Which kind of autonomic algorithms should we enforce to add robustness and self-organization properties to those systems?

Our future research within the CASCADAS European project will try to address some of these questions.

References

[Bal06] H. Balakrishnan, S. Madden, V. Bychkovsky, K. Chen, W. Daher, M. Goraczko, H. Hu, B. Hull, A. Miu, E. Shih, "CarTel: A Mobile Sensor Computing System", MIT CSAIL Report, 2006.

[BeaB06] J. Beal, J. Bachrach, "Infrastructure for Engineered Emergence on Sensor/Actuator Networks", IEEE Intelligent Systems, 21(2):10-19,2006

[BecH04] C. Becker, M. Handte, G. Schiele, K. Rothermel, "PCOM - A Component System for Pervasive Computing", IEEE International Conference on Pervasive Computing and Communications, Orlando (FL) ,USA , 2004.

[Bor04] C. Borcea, D. Iyer, P. Kang, A. Saxena, L. Iftode, "Spatial Programming using Smart Messages: Design and Implementation", International Conference on Distributed Computing Systems, Tokio, Japan, 2004.

[Bor05] G. Borriello, "RFID: Tagging the World", Communication of the ACM, ACM Press, 48(9):34-79, 2005.

[But06] D. Butler, "2020 Computing: everything, everywhere", news@nature.com, 22 March 2006.

[Cas06] G. Castelli, M. Mamei, A. Rosi, F. Zambonelli, "Browsing the World: bridging pervasive computing and the Web", International Workshop on Ubiquitous Geographical Information Services, Munster, Germany, 2006.

[Chu06] C. Park, P. Chou, Y. Sun, "A Wearable Wireless Sensor Platform for Interactive Art Performance", IEEE International Conference on Pervasive Computing and Communications, Pisa, Italy, 2006.

[CurG05] C. Curino, M. Giani, M. Giorgetta, A. Giusti, A. Murphy, G. Picco, "TinyLIME: Bridging Mobile and Sensor Networks through Middleware". IEEE International Conference on Pervasive Computing and Communications, Kauai Island, HW (USA), 2005.

[Dan06] W. Daniel, T. Pintaric, F. Ledermann, S. Dieter, "Towards Massively Multi-User Augmented Reality on Handheld Devices", International Conference on Pervasive Computing, Munich, Germany, 2005.

[Dob07] S. Dobson, S. Denasiz, A. Fernandez, D. Gaiti, E. Gelenbe, F. Massacci, P. Nixon, F. Saffre, N. Schmidt, F. Zambonelli, "A Survey of Autonomic Communications", ACM Transactions on Autonomous and Adaptive Systems, ACM Press, to appear, 2007.

[Eag03] N. Eagle, P. Singh, and A. Pentland, "Common sense conversations: understanding casual conversation using a common sense database". Workshop on Artificial Intelligence, Information Access, and Mobile Computing, Acapulco, Mexico, 2003

[Eag05] N. Eagle and A. Pentland, "Social Serendipity: Mobilizing Social Software", IEEE Pervasive Computing, IEEE CS Press, 4(2):28-34, 2005.

[Esp01] F. Espinoza, P. Persson, A. Sandin, H. Nystrom, E. Cacciatore, M. Bylund, "GeoNotes: Social and Navigational Aspects of Location-Based Information Systems", International Conference on Ubiquitous Computing, Atlanta (GE), USA, 2001.

[HadM06] S. Hadim, N. Mohamed,"Middleware Challenges and Approaches for Wireless Sensor Networks", IEEE Distributed Systems Online, 7(3), 2006.

[HarK05] R. Hariharan, J. Krumm, E. Horvitz, "Web-Enhanced GPS", International Workshop on Location and Context Awareness, Munich, Germany, 2005.

[HigB01] J. Hightower, G. Borriello, "Location Systems for Ubiquitous Computing", IEEE Computer, IEEE CS Press, 34(8):57-66, 2001.

[IntG00]C. Intanagonwiwat, R. Govindan and D. Estrin, "Directed diffusion: A scalable and robust communication paradigm for sensor networks", International Conference on Mobile Computing and Networking, Rome, Italy, 2000.

[Jel05] M. Jelasity, A. Montresor, O. Babaoglu, "Gossip-Based Aggregation in Large Dynamic Networks", ACM Transaction on Computer Systems, ACM Press, 23(3):219-252, 2005.

[Jia06] M. Jia, X. Fan, X. Xie, M. Li, W. Ma, "Photo-to-Search: Using Camera Phones to Inquire of the Surrounding World", International Conference on Mobile Data Management, Nara, Japan, 2006.

[Jon01] C. Jones, K. Sivalingam, P. Agrawal, J. Chen, "A Survey of Energy Efficient Network Protocols for Wireless Networks", Wireless Networks, Kluwer Academic Publisher, 7(4):343-358, 2001.

[JonG05] Q. Jones, S. Grandhi, "P3 Systems: Putting the Place Back into Social Networks", IEEE Internet Computing, IEEE CS Press, 9(5):38-46, 2005.

[JRDMS] James Reserve Data Management System, http://dms.jamesreserve.edu

[LegT06] C. Legner, F. Thiesse, "RFID-Based Facility Maintenance at Frankfurt Airport", IEEE Pervasive Computer, IEEE CS Press, 5(1):34-39, 2006.

[Liu04] H. Liu, P. Singh, "ConceptNet: A Practical Commonsense Reasoning Toolkit", BT Technology Journal, Kluwer Academic Publishers, 22(4):211-226, 2004.

[MadF02] S. Madden, M. Franklin, J. Hellerstein, W. Hong. "TAG: A Tiny AGgregation Service for Ad-Hoc Sensor Networks", Symposium on Operatign Systems Design and Implementation, Boston (MA) ,USA, 2002.

[MamQZ06] M. Mamei, R. Quaglieri, F. Zambonelli, "Making Tuple Spaces Physical with RFID Tags", ACM Symposium on Applied Computing, Dijon, France, 2006.

[MamZ05] M. Mamei, F. Zambonelli, "Physical Deployment of Digital Pheromones Through RFID Technology", IEEE Swarm Intelligence Symposium, Pasadena (CA), USA, 2005.

[Nag02] R. Nagpal, "Programmable Self-Assembly Using Biologically-Inspired Multiagent Control", Joint Conference on Autonomous Agents and Multi-Agent Systems, Bologna, Italy, 2002.

[NagM04] R. Nagpal, M. Mamei, "Engineering Amorphous Computing Systems", Methodologies and Software Engineering for Agent Systems, Kluwer Academic Publishing, 2004.

[NatR06] B. Nath, F. Reynolds, R. Want, "RFID Technology and Appliations", IEEE Pervasive Computing 5(1):22-69, 2006.

[NewA05] R. Newton, Arvind, M. Welsh, "Building up to Macroprogramming: An Intermediate Language for Sensor Networks", International Symposium on Information Processing in Sensor Networks, Los Angeles (CA), USA, 2005.

[PatL04] D. Patterson, L. Liao, K. Gajos, M. Collier, N. Livic, K. Olson, S. Wang, D. Fox, H. Kautz, "Opportunity Knocks: a System to Provide Cognitive Assistance with Transportation Services", International Conference on Ubiquitous Computing, Tokyo, Japan, 2004.

[PerP04] M. Perkowitz, M. Philipose, D. Patterson, K. Fishkin. "Mining Models of Human Activities from the Web". International World Wide Web Conference, New York (NY), USA, 2004.

[Pol06] J. Polastre, R. Szewcyk, A. Mainwaring, D. Culler, J. Anderson, "Analysis of Wireless Sensor Networks for Habitat Monitoring", Wireless Sensor Networks, Kluwer Academic Publishers, 2004.

[Rie06] M. Rieback, B Crispo, A. Tanenbaum, "Is Your Cat Infected with a Computer Virus?", IEEE International Conference on Pervasive Computing and Communications, Pisa, Italy, 2006.

[Sat05] I. Satoh, "A Location Model for Pervasive Computing Environments", Proceedings of IEEE International Conference on Pervasive Computing and Communications, Kauai Island, HW (USA), 2005.

[ShrP04] Nisheeth Shrivastava, Chiranjeeb Buragohain, Divyakant Agrawal and Subhash Suri, "Medians and Beyond: New Aggregation Techniques for Sensor Networks," ACM Conference on Embedded Networked Sensor Systems (SenSys 2004), 2004.

[Sri06] M. Srivastava, M. Hansen, J. Burke, A. Parker, S. Reddy, G. Saurabh, M. Allman, V. Paxson, D. Estrin, "Wireless Urban Sensing Systems", CENS Technical Report #65 , 2006.

[StoN04] K. Stoy, R. Nagpal, "Self-Reconfiguration Using Directed Growth", International Symposium on Distributed Autonomous Robotic Systems", Toulouse, France, 2004.

[Wan04] R. Want, "Enabling Ubiquitous Sensing with RFID", IEEE Computer, IEEE CS Press, 37(4):84-86, 2004.

[Wan06] R. Want, "An Introduction to RFID Technology", IEEE Pervasive Computer, IEEE CS Press, 5(1):25-33, 2006.

[WelN04] M. Welsh, R. Newton, "Region streams: functional macroprogramming for sensor networks", International Workshop on Data Management for Sensor Networks, Toronto, Canada, 2004.

[WerB06] J. Werfel, Y. Bar-Yam, D. Rus, R. Nagpal, "Distributed construction by mobile robots with enhanced building blocks", International Conference on Robotics and Automation, Orlando (FL), USA, 2006.

[WerL06] G. Werner-Allen, K. Lorincz, M. Ruiz, O. Marcillo, J. Johnson, J. Lees, M. Welsh, "Deploying a Wireless Sensor Network on an Active Volcano.", IEEE Internet Computing, IEEE CS Press, 10(2):18-25, 2006.

[YaoG02] Y. Yao, J. Gehrke, "The Cougar approach to in-network query processing in sensor networks", ACM Sigmod Record, ACM Press, 31(3):9-19, 2002.

[Zam04]F. Zambonelli, V. Parunak, "Towards a Paradigm Change in Computer Science and Software Engineering: a Synthesis", The Knowledge Engineering Review, 18(4), 2004.

Towards an Agent Model for Future Autonomic Communications

Francesco De Mola[1], Raffaele Quitadamo[1]

Abstract — **The continuous growth in ubiquitous and mobile network connectivity, together with the increasing number of networked computational devices populating our everyday environments (e.g., PDAs, sensor networks, tags, etc.), call for a deep rethinking of traditional communication and service architectures. The emerging area of autonomic communication addresses such challenging issues by trying to identify novel flexible network architectures, and by conceiving novel conceptual and practical tools for the design, development, and execution of "autonomic" (i.e., self-organizing, self-adaptive and context-aware) communication services. In this paper, after having introduced the general concepts behind autonomic communications and autonomic communication services, we analyze the key issues related to the identification of suitable "component" models for autonomic communication services, and discuss the strict relation between such models and agent models. On this basis, we try to synthesize the key desirable characteristics that one should expect from a general-purpose agent model for autonomic communication services.**

Index Terms— **Autonomic Communication, Services, Self-organization, Self-adaptation, Multiagent Systems**

I. INTRODUCTION

OUR everyday world is increasingly being populated with a wide variety of new communication technologies and computing devices. On the one hand, several wireless and ad-hoc communication solutions are being deployed with the potential of ensuring us 24/7 ubiquitous connectivity to the Internet and to the surrounding devices. On the other hand, devices such as sensor networks [6], RFID tags [18], cameras, GPS and other location systems [9], will enable us to dynamically acquire information and interact with the physical world.

The above scenario opens up the possibility for a wide range of brand new applications (e.g., on-line monitoring of the world [6] and enhanced social experiences [13]), as well as for enhancing the quality and effectiveness of current communication services (via context-awareness and dynamic personalization [5]). However, it also introduces a dramatic increase in complexity and a number of novel design issues, challenging current communication and distributed computing paradigms, and making it difficult to deliver the promised benefits in truly usable and economically feasible ways.

The complexity we are talking about is due to several factors, there included:

- *Heterogeneity of involved components*. The range of newly introduced network and computing technologies is already wide and it is expected to grow consistently.
- *Dynamism of network scenarios and applications.* As connectivity is becoming ubiquitous and mobile, and as computers are getting embedded in our everyday objects, the resulting network becomes highly dynamic in terms of topology and usage patterns.
- *Decentralization and unreliability*. The highly decentralized and embedded nature of the involved components, makes it hard (whether not impossible) to enforce some forms of direct control over their configuration and their activities.

Other than from the above sources of complexity, additional challenges are introduced by the need of exploiting in full the potentials of the new scenarios and put them at the service of users. This implies identifying suitable models and tools by which *innovative services* can be designed, developed and deployed, and by which existing and new services can be made more flexible and *dynamically adaptable*, i.e. able to properly react to the dynamics and unreliability of the scenario without suffering from any malfunctioning, and able to increase user satisfaction by adapting their behaviour to the current context (physical and/or social) of users and to their own individual needs.

The emerging inter-disciplinary research area of autonomic communication [1, 14] attempts to overcome the limitations of current communication models and architectures in addressing complexities and issues raised by modern network scenarios. In particular, autonomic communication broadly relates to the study and development of novel semantic communication models [5], novel adaptive and evolvable architecture for network components [3], as well as novel paradigms and tools for the design, development, and execution of *autonomic* (i.e., self-organizing, self-adapting, and context-aware) *communication services* [12].

[1] Dipartimento di Ingegneria dell'Informazione – Università di Modena e Reggio Emilia - 41100 Modena, Italy, email: {demola.francesco, quitadamo.raffaele}@unimore.it

In this paper, we specifically focus on autonomic communication services with the goals of: *(i)* analysing the key issues related to the identification of novel software engineering approaches and of a novel "component" model for the design and development of autonomic communication services (Section II); *(ii)* eventually, trying to synthesize the key desirable characteristics that one should expect from a general-purpose component model for autonomic communication services and the contributions that can come from the agent community (Section III). The key message we hope to get home is that current researches in software agents and multi-agent systems have the potential for playing a major role in inspiring and driving the identification of such model, and more in general for influencing and advancing the whole area of autonomic communication.

II. Autonomic Communication Services

Autonomic communication generally refers to all those research thrusts involved in a deep foundational re-thinking of communication, networking, and distributed computing paradigms, to face the increasing complexities and dynamics of modern network scenarios. The ultimate vision of autonomic communication researches is that of a networked world, in which networks and associated devices and services will be able to work in a totally unsupervised – i.e., autonomic [10] – way, being able to self-configure, self-monitor, self-adapt, and self-heal. To some extent, the idea is to consider networks as sorts of immense organisms, and by conceiving components within as parts of these organisms, able to prosper and autonomously survive contingencies [12]. On the one hand, this will enable to effectively have networks capable of dynamically adapting their behaviour to meet the specific needs of individual users. On the other hand, this will enable to dramatically decrease the complexity – and the associated costs – currently involved in the effective and reliable deployment of networks and communication services.

A. *Scenarios of Autonomic Communication*

The need for re-thinking communication and distributed computing paradigms directly derives from the novel characteristics exhibited by modern and emerging network scenarios. Traditional communication and distributed computing paradigm were conceived to target a now obsolete perspective of computer networks: wired networks of (rather homogeneous) medium/high-end computers and routers. In such scenarios, network disconnections and failure of components are considered exceptions, and network and system managers are always assumed to be able to act on the system for re-configuration and fault-recovery. However, as stated in the introduction, modern network scenarios more and more include a large number of very heterogeneous components (from low-end computer-based sensors, to PDA, laptops, and workstations), interacting over a variety of wireless channel (from WiFi, to Bluetooth and ZigBee), and in the presence of mobility (of both devices and users exploiting them). There, failures of components and network disconnections are the norm rather than the exception, and the possibility for network and system managers to intervene in the system is challenged by the intrinsic decentralization and complexity of the scenario.

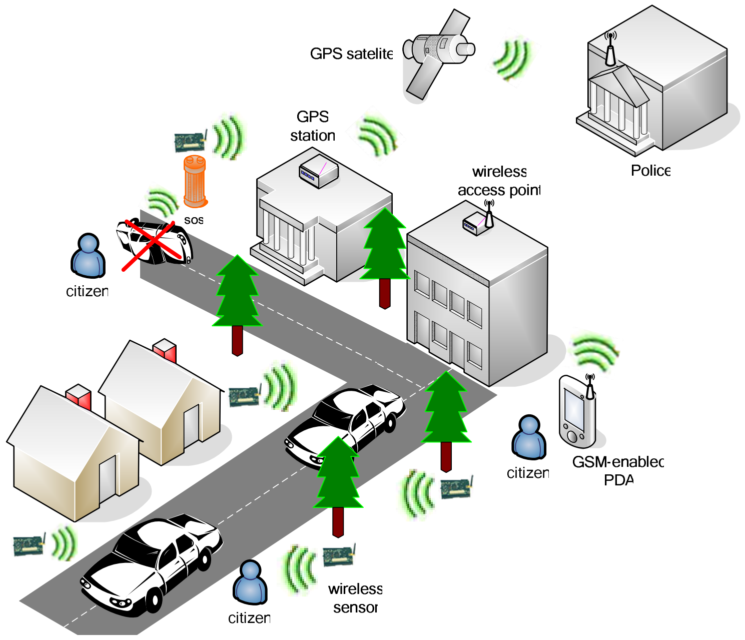


Figure 1. An urban scenario of autonomic communication

Just to reach a better understanding of what such scenarios could look like, imagine what our cities will be in the next few years (see Figure 1). First, a variety of computer-based sensors will be spread around in every street, crossing, squares, and within buildings. We can already find a variety of simple sensors around in our cities (e.g., to measure traffic intensity and pollution), but the future will see these sensors become wireless-enabled, and dramatically increase in density and diversity. It is not unrealistic to think that – say in ten years – it will be possible to determine in real time how many free benches are there in a specific park or how long is the queue at the nearest post office. Second, wireless-enabled computing devices will be worn by each and every person and will be embedded in any cars. Such devices, beside the capability to access to the Internet, will also be able (via ad-hoc wireless communications) to directly interact with each other and with sensors around, and localize themselves via GPS or other means. After all, smart phones with GPS, Bluetooth, and cameras, are already a reality. Third, all of these devices will be able to mobilize data from and to the Internet, based on a variety of communication channels, from WiFi, to UMTS or satellite communications.

The heterogeneity of components and network technologies involved in the above scenario is very evident, as it is the fact that the resulting network is highly dynamic (due to both the unreliable nature of sensors and the ephemeral and mobile nature of wearable and car-embedded devices) and highly decentralized (no system manager could enforce a strict control over dispersed sensors and over personal devices). This factors clearly justify the efforts of autonomic communication researches towards the identification of: (i) innovative ways of modeling communication, suitable for dense dynamic networks of wireless devices and overcoming the limitations of traditional point-to-point Shannon-oriented communication models; (ii) the definition of innovative flexible architecture for network devices, suitable to tackle dynamics and decentralization via dynamic re-configuration; (iii) the identification of innovative models and tools for the design, development, and execution of autonomic communication services.

B. *Towards Autonomic Communication Services*

In general terms, a communication service is a functionality that is made available within a network to access and exploit the network resources. IP datagram routing, DNS, socket-based point-to-point communication, cryptographic tools, web services and P2P data delivery services, can all be considered communication services. The definition applies independently of the fact that such services can be conceived to act either as "user-level" services or as "infrastructural" ones, to be put at the service of other services.

In the sketched scenario, a variety of communication services can (should) be put in place to properly access and exploit the available network and computing resources, some of which of an innovative nature.

At the more infrastructural level, we could think at localization services that, by exploiting GPS, WiFi signal strength, or whatever localization tool is available, are able to provide the location of users, cars, or devices. Also, we could think of a variety of routing services, able to deliver data and messages across the network, from more traditional routing services offering delivery to a specific network ID, to more advanced routing services capable of delivering messages at specific locations of the network (for which the routing service has to exploit the available localization services) or to multicast messages at specific groups of nodes or users.

Shifting to the user level, the presence of sensors, ubiquitous and mobile connectivity, localization services, opens up an incredible range of possibilities for the deployment of highly innovative and useful services. By properly exploiting sensors, localization services, and proper routing services, one could think of making available to users various services to query the physical world and obtain any kind of information about the surrounding situation (there included other users), and possibly to integrate this information with information dynamically downloaded form the Web. As another example, one could think at elaborated services to alleviate roads congestion problems. This would imply devices in cars (for computing, sensing and visualization) to interact with devices in streets and crossings (for sensing the current traffic situation and communicate it to cars). Cars could also interact with each other (the same as sensor could) and form wireless ad-hoc network that can be used to properly forward information across the town. The overall service could then exploit all this available information to map in real-time the status of streets in the city, and calculate on-the-fly faster routes for users that avoid congestion areas or areas that are likely to become congested soon.

Whatever communication service one can think of, and whether at the infrastructure level or at the user level, it is clear this will be generally realized in terms of some software components (though it may be the case that some components can be directly encoded in hardware) [11, 12]. Such components will act as access points to the service, and will be able to provide the service either in autonomy or by interacting with each other in the distributed network environment, as it should necessarily happen for all those services, like routing, which are of an intrinsic distributed nature.

However, in modern network scenarios, the only possible way to effectively develop and deploy services is by making them autonomic, (i.e., capable of self-organization, self-monitoring, self-healing) and flexibly adaptable (to meet very diverse situations and diverse user needs). For instance, a localization service should always be able to provide information on a best-effort basis in any situation, without rigidly requiring the availability of specific localization devices and rather exploiting a variety of heterogeneous localization devices. As another example, a routing service should guarantee message delivery in very dynamic and mobile networks, without requiring manual reconfigurations, and should possibly tune quality-of-service depending on the specific needs of the user/application exploiting it. These needs induce specific requirements on what a proper autonomic and adaptive component model for autonomic communication services should be, and also forces abandoning traditional (i.e. stack layered) communication service architectures.

C. *Requirements for Autonomic Communication Services*

The need for communication services to fit the complexities of modern network scenarios by becoming autonomic and adaptable, calls for an underlying component model capable of satisfying a set of requirements. In particular:

- *Autonomicity*. A component model for autonomic communication services implies the capability of components (at the individual level, or at an aggregate social level, or both) to support self-preservation and self-healing of some specific functional and/or non-functional properties independently of contingencies, just like a living organism is able to maintain its internal balances [10].
- *Dynamic Adaptation to Changes*. A component model requires the capability of tolerating dynamic self-reconfiguration of components, and of their composition and interaction patterns, without requiring any a priori information and/or human intervention.
- *Situation-Awareness*. To achieve autonomic behaviour and adaptivity, a component model for autonomic communication services must be necessarily aware of what's happening around.
- *Generality*. It is expected that next-general autonomic communication services will involve several components executing on a variety of heterogeneous devices and interacting via a variety of communication technologies.
- *Scalability*. Given the possible very large scale of the target network scenarios, the component model should be based on design principles that can be

practically applicable to small systems as well as very large systems, and should promote organizing services according to patterns that exhibit scalable performances (or quality of service).

At this point of the discussion, the reader will probably already think that the requirements for such envisioned component model can simply and directly be mapped into an agent-based model, and that the ecology of autonomic communication services can be considered as a sort of complex agent society. This is true only to some extent and the rest of the paper will better unfold and analyze such an issue, keeping in mind the above mentioned requirements throughout the discussion.

III. TOWARDS A MODEL FOR AUTONOMIC COMMUNICATION SERVICE AGENTS

In this section, we claim that an agent model can be the most suitable answer to the challenging requirements of autonomic communications. Nonetheless, past agent models do not fulfill all the requirements discussed earlier and thus we stated that such a model should exhibit some peculiar features that we try to discuss in the remainder of the paper.

A. Agents as Autonomic Service Providers

Taking into account the intrinsic dynamicity and complexity of the above scenario and its requirements, it clearly emerges that autonomic communication services cannot be modeled and implemented as "passive components", like in a standard service-oriented architecture. Rather, autonomic communication services should be modeled and implemented by "active" autonomous components, exposing their service and integrating (at the component or at the system level) features of autonomicity, self-adaptation, and situation-awareness, in a scalable and general way. Accordingly, at this point, we can state that the search for a novel autonomic component model for autonomic communication services corresponds to the search for a proper "service agent" model.

In general, we envision that the nodes of an autonomic communication architecture should host some sort of *agent/service execution environment* on top of the operating system (see Figure 2), to act as a general flexible support for the execution of service agents. The execution environment should tolerate the hosting of both very simple reactive agents and of more heavy-weight "intelligent" self-adaptive agents. Furthermore, it is likely that such environment will have to host also other kinds of "artifacts", such as tuple spaces, resources, channels and so forth. The execution environment should be as *thin* as possible: it should provide only the minimal set of basic services to agents (e.g., agent creation and cloning, capability to perceive local events), so as to make it possible to run it even on small resource-constrained devices, like sensors or smart-phones. Upon the distributed set of execution environments, agents of different types can execute, reproduce themselves, and interact with each other. Whenever a specific autonomic communication service is needed, users (or other agents) can provide it, "injecting" the proper service agents in the network. Any type of communication service, from infrastructural ones to user-level ones, is realized by specific service agents deployed in the infrastructure, without any pre-defined "layering". Rather, the idea is that of an "ecology" of distributed agents, in which different species of agents, from ant-based to intelligent ones co-exist, each providing specific services either as a species or as individuals, and interacting with each other so as to gather what services they need from each other.

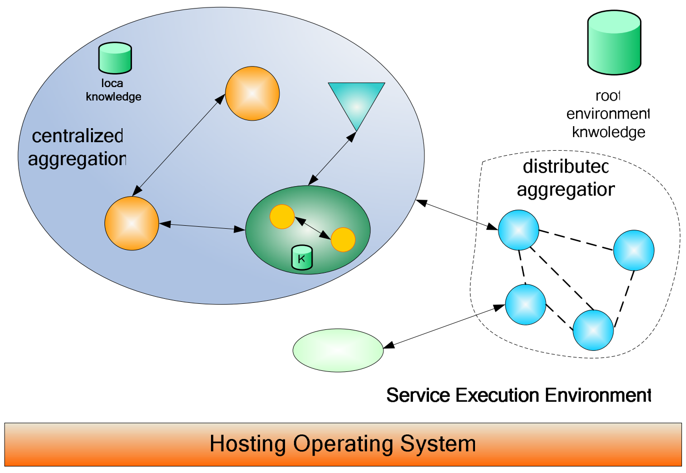


Figure 2. Aggregation of autonomic service agents

In this general scenario, satisfying the requirements of autonomicity, self-adaptability, situation-awareness, scalability, and generality, requires defining a service model and associated tools to support:

- different forms of spontaneous *self-aggregation* by service agents, to enable both multiple distributed agents to collectively and adaptively provide a distributed communication service and a service agent to properly exploit other services on need;
- some ways to *enforce control* in the ecology of service agents;
- *self-similar* forms of aggregation, capable of reproducing nearly identical structures over multiple scales, and achieving software engineering scalability;
- suitable models for the *organization of situational* information and their access by agents, promoting more informed adaptation choices by agents and advanced forms of stigmergic interactions.

These issues are analyzed in the following of this section.

B. Self-Aggregation as an Adaptation Mechanism

In an "ecology" of self-adaptive service agents executing on a very thin and bare environment, *self-aggregation* is the key mechanism to build and exploit complex communication services. Self-aggregation is clearly an autonomic adaptation process, in that it must occur on need and without direct human intervention: whenever some changes occur in the surrounding environment, some simple communication services can decide to form a coalition that can better handle the new unforeseen situation or provide an improved service.

Enabling self-aggregation in our agent model implies rethinking traditional integration architectures both from an *architectural* and a *behavioural point of view*.

Let's consider, first of all, the *architectural* viewpoint, i.e. how our service agents should be designed to support aggregation formations and to accomplish the discussed requirements of autonomic communications. A TCP socket can be seen, for instance, as a composition of layered services, e.g. the IP routing service and the Ethernet data link service. This type of composition can be defined a *containment*, because an outer component (i.e. the TCP service) encapsulates one or more inner components (i.e. the IP service) and uses their services. Every service request delivered to the outer component is forwarded to an inner one and, while negligible in many simple aggregations, the overhead introduced by this forwarding can be significant in resource-constrained devices and/or when more complex aggregations are formed. Autonomic communication services overcome the limits of layering, proposing a flattened model, in which composing service agents means rather *combining/fusing their service interfaces into a new negotiated interface*. This avoids the overhead of forwarding messages/function calls across the several layers of the aggregation.

Moreover, our flattened service model means also that agents should be allowed to participate in more than one aggregation, e.g. because their service can be shared among different clients. This requires that our service agent should be more than simply a "service provider" with a fixed published interface. We envision a new concept of *interface* that is far more flexible than classical software component interfaces. When a service agent participates in an aggregation, its interface should be updated or, better, it must expose a new interface: the "aggregate service interface". Such interface is expected to be the same in every agent participating in that aggregation and its provided operations are negotiated and constructed according to the aggregation strategy and the requirements of the new complex service. As a consequence, in our agent model service agents are considered sort of "polyhedral components", capable of exposing several interfaces as different service access points. Let's consider, for example, the case of a service agent participating in more than one aggregation: it has to dynamically choose the right interface to expose, depending on the access point from where it is accessed.

Moreover, each provided interface can even change depending on internal reorganizations of the aggregate service, e.g. due to adaptation to environmental happenings, like the failure of one aggregated service. Handling multiple and dynamic interfaces requires a sort of *interface negotiation mechanism* among service agents: in place of fixed interfaces, the interface negotiation mechanism defines and requires service agents to support universally known "introspection facilities" by which support for other services can be ascertained at runtime. Clients of a service agent use these well known services to obtain mutually agreeable interfaces.

Autonomic communication services are very often located on different network nodes and the aspect of the *physical distribution* of services should be taken into account, even though the aggregated service appears *conceptually unique*. For example, services provided by self-organizing swarm agents are usually distributed by their own nature (e.g. some localization service whose agents are distributed across many sensors in the environment), and do not respond to a central controller or supervisor; nonetheless, these agents should ideally work as a unique service, which can be invoked from many scattered access points. This leads as a consequence that our agent model should transparently support "at least" both centralized and distributed service aggregations (see Figure 2) and we said "at least", because there might be intermediate or hybrid solutions between these two extremes. *Centralized aggregations* are those where many service agents, locally available or instantiated at runtime, are combined into a new complex service. In this case, the "aggregator agent" is the new centralized access point to the service and is not physically distributed. It exposes a single communication interface to other services all over the network. *Distributed aggregation* raises more challenging design issues, because, in this case, several agents decide to join together into an aggregated service, but they still preserve their physical distribution in the network. In other words, they all agree on a common "aggregate service interface", but there is no aggregator agent exposing it; every single participant instead is considered "access point" to the aggregate service and exposes the same interface as all others.

Besides architectural design choices, self-aggregation needs effective algorithms and tools to work in dynamic and open environments, without human intervention. From a more *behavioural* standpoint, service agents are expected to support different aggregation techniques, which are an active research area of AI. Several coalition formation algorithms have been proposed for task allocation problems [16, 15] and, although we are not interested here in one particular algorithm, we state that autonomic self-aggregation will likely draw much inspiration from such research work. Therefore, each service agent in our model must include a proper *aggregation interface* (through which the agent can be involved in new aggregations, leave broken coalitions and so on) and such interface should be as much general as possible, to support a wide range of coalition formation algorithms. Finally, we must recall that autonomicity should be enforced at all levels of aggregation and this requires proper mechanisms to control/supervise the behaviour of the aggregated components. Such issues are the subject of the next Subsection.

C. *Enforcing Control for Self-management*

As already highlighted, one of the key driving principle of the autonomic communication vision is that services should be *self-managing*. The fundamental problem when trying to enable autonomicity (at all levels of service aggregation, from primitive service to complex ones) consists in *establishing some kind of control over service agents*, in order to constantly guarantee an optimal overall functionality, protect against malfunctioning parts and so forth. The IBM proposal for building autonomic components [10] is based upon the introduction of the so-called "autonomic manager" (see Figure 3), which is an intelligent software entity that monitors the activity of its managed resource, and can take corrective actions in a sort of continuous control loop. Nonetheless, control and supervision at the individual level does not guarantee an autonomic behaviour of the entire system: in an aggregation of service agents, where every member constantly monitors and regulates its own essential variables (i.e. in a local loop), it often occurs that the selfish nature of each component does not result in an optimal outcome of the aggregated service. Applied to our highly dynamic, open and distributed scenarios, the problem of enforcing control and achieving an adequate level of self-management is even trickier.

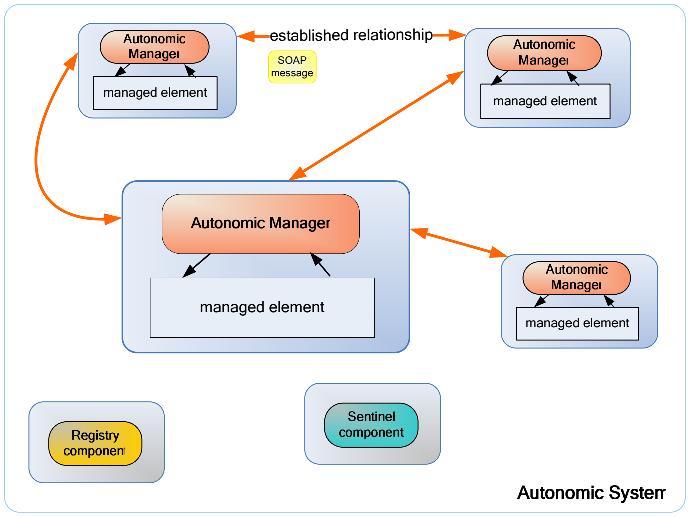


Figure 3. Autonomic components in the IBM perspective

To this purpose, some more traditional design patterns introduce a special "sentinel element", in charge of supervising the behaviour of each autonomic element and avoid dangerous or incorrect actions: many autonomic component frameworks (e.g. [11]) adopt this pattern, since they continuously monitor deployed components and influence their behaviour injecting proper "adaptation rules". These rules are interpreted by the component and translated in corrective actions on its internal parameters, leading hereafter to a modified behaviour of the same component. Other more agent-inspired solutions rely on the "cooperation capability and sociality" of autonomic managers: exchanging information with each other and orchestrating their actions, these intelligent individual controllers can ensure an *autonomic behaviour* of a composed service. All such approaches are essentially coupling traditional monitoring and resource management with artificial intelligence techniques for planning and knowledge management, as well as multi-agent systems negotiation ones.

Nevertheless, we argue that these approaches, though still feasible and valid, will prove to be increasingly unsuitable for many autonomic communication scenarios, like they have been presented so far. Autonomic communication services are expected to be pervasive and to run on even small wearable devices and, having autonomic managers logically separated from their managed services, can produce heavyweight service agents. In fact, designing each single service agent, with the rational capability to react to all possible contingencies planning proper corrective actions, may end up in a cumbersome service architecture. It can be stated that the limitations of the discussed autonomic self-management derives from its being inspired by traditional human-based management, where we usually have the controller and the controlled entity. Self-organization approaches support instead the biologically inspired idea that "a system should be able to self-manage by its own very nature and not by external intervention of other "non-self" entities" [19]. Self-organizing systems exhibit an intrinsic self-managing capability, which only indirectly depends on the behaviour of their individual little agents, but rather descends for the combination of their local interactions. Service emergence helps avoiding intelligent elements, with planning and knowledge management capabilities to react to unforeseen environmental changes, and produces simpler and more lightweight architectures. As autonomic computing design patterns have their drawbacks, current approaches to self-organization are likewise limited, e.g. because they can implement only a limited set of self-managing functionalities, but often fail in accounting the diverse and complex requirements of autonomic communications.

The agent model we are sketching in this paper should be thus as much flexible and general-purpose as possible. It should still allow both traditional autonomic managers and self-organizing approaches, but here we deem crucial to introduce also an innovative vision of self-management [19] tailored to the peculiarities of autonomic communications. Since most autonomic communication usage scenarios will be dramatically distributed, often without any clearly identifiable stakeholders, the only solution to enforce some forms of control over them, and to have the self-management features of each individual system coexist with more decentralized forms of self-management, will be that of populating the ecosystem with additional "manager components". In an environment where every single service, even the most basic one, is provided by a service agent, it is reasonable to assume that self-management should be enforced by means of some first-class elements, injected on demand into the self-organizing system. These "manager agents" will have to live inside the system and interact with other self-organizing service agents, to monitor their execution and possibly influence their emergent behaviour. This brings as a consequence that the knowledge management and planning capability, previously placed as a possibly heavyweight burden on every single component, is now "externalized" and made distributed across the various deployed manager agents. It must be pointed out that some of these ideas have been already experimented and formalized in MAS research: the idea of Electronic Institutions (EI) and *norm-aware agent societies* have been proposed as a model to specify the kinds of interactions among software agents using norms (e.g. obligations, permissions, etc.). In [2] norms are explicitly represented and managed via rules and a team of "administrative (institutional) agents" is deployed in the distributed architecture, to ensure normative positions are complied with and updated by individual agents. Experiences from this and other research on norm-based systems will be of paramount importance to formalize our "ecology-like" autonomic communications service model.

D. *Robustness and Generality with Holonic Agents*

Given the importance of self-aggregation in our model, the combination of primitive services into new complex ensembles must be fully *scalable*, i.e. the software design principles should be applicable to small systems, as well as very large systems, possibly made by huge numbers of heterogeneous nodes and service components. In our service agent model, all agents should at least expose a common set of basic functionalities, i.e. a "common interface", besides their specific peculiar operations. Aggregate service agents will be services in their turn and will thus have the same basic shared interface. Applying the *self-similarity* principle means that "individual components self-organize and self-aggregate so as to reproduce nearly identical structures over multiple scales" [4].

From a software engineering point of view, having the same structural and organizational principles in force at different scales facilitates the management of services: e.g. if service agent A decides to aggregate with service agent B, it can at least rely upon the shared common interface to negotiate the aggregation and agree on the new aggregated interface to expose. Self-similarity helps to achieve the key requirement of *generality*: this feature is fundamental to better handle design complexity in an environment where possibly thousands of heterogeneous software agents can be hosted. It would allow "diving" into specific sub-systems whenever necessary, without having to modify abstractions and tools to work at finer levels of granularity.

From a more architectural standpoint, self-similar structures are known to be intrinsically *robust*: it is more than desirable that the combination of some autonomic communication services brings to entities that are robust and capable of adapting to changes in the environment (e.g. a wireless link goes down, but the service will find alternative paths to deliver the message). Many biological systems exhibit such properties, thanks to their being organized in hierarchical and *self-similar* structures at different scales.

A successful agent model for autonomic communication services should therefore support self-similar aggregation and MAS research has already explored some important applications in this direction, introducing the promising idea of *holonic agents* [7, 8] (mainly applied to manufacturing scenarios). The term *holon* was originally introduced by the philosopher Arthur Koestler in order to name recursive and self-similar structures in biological and sociological entities: a lot of systems in nature can be seen as either "whole" or "part" of a larger system; for example, a human individual is on the one hand a composition of organs, consisting of cells that can be further decomposed, and on the other hand he (or she) may be part of a group which in turn is part of the human society. According to Koestler, a holon is a structure that is stable and coherent and that consists of further holons that function similarly. Koestler defines a *holarchy* as a hierarchy of self-regulating holons which function

a. as autonomous wholes in supra-ordination to their parts,
b. as dependent parts in sub-ordination to controls on higher levels,
c. in co-ordination with their local environment.

Therefore, it is clear how *self-similar service aggregations* are endowed with the properties that are intrinsic in this holarchy definition. Building holonic service compositions enables the construction of very complex systems that are efficient in the use of resources, highly resilient to disturbance (both internal and external) and adaptable to changes in their surrounding environment. Holarchies (i.e. service aggregations) are *recursive* in the sense that a holon (i.e. an agent) may itself be an entire holarchy that acts as an autonomous and cooperative unit in the first holarchy. The *stability* of holonic service aggregations stems from holons being self-reliant units, which *have a degree of independence and handle circumstances and problems on their particular level of existence* (i.e. the local execution environment of the aggregator agent), without asking higher level holons for assistance. Holons can also receive instructions from and, to a certain extent, be controlled by higher level holons. The self-reliant characteristic ensures that holons are able to survive disturbance, while the subordination to higher level holons ensures the effective operation of the larger whole.

Like holons, self-similar aggregate agents would participate in further aggregations/holarchies or would simply exist as new available services, but always as self-reliant units: hiding their internal complexity under a self-similar interface, they can react to changes in the environment and *adapt* to different situations, transparently re-organizing their internal structure.

E. *Organizing Situational Data into Knowledge Networks*

Another essential requirement for autonomic communication services is their capability to perceive their surrounding context and consequently adapt and improve their behaviour. Information about the context is expected to be increasingly important to enable *situation-awareness* in next generation communication services.

Nowadays, several mechanisms exist to produce situational data from the environment (e.g. intelligent sensors or monitoring mechanisms) and such knowledge is expected to become a dramatic amount in the near future. In our vision, this huge amount of information cannot be fully managed or internalized by every single service agent: it would require a significant knowledge management capability that we consider an avoidable burden in our agent model. Our basic idea is that situational data should be somehow scattered part in the environment (e.g. in a shared tuple space) and part across the different service agents. In further details, we envision that when service agents decide to form aggregations, they share their pieces of context knowledge with the other participants, forming a sort of "aggregated situational knowledge". This knowledge, scattered among aggregate agents, will be thus organized in a hierarchical fashion among all the running service agents: in a few words, one agent could own a piece of knowledge about the local context and, by joining an aggregation, it would integrate its information within the aggregated knowledge. The aggregated entity, being self-similarly part of another aggregate or of the Service Execution

Environment (see Figure 2), would perform the same integration in turn. Therefore, the global knowledge would be dynamically built by the various service agents that join and leave the system during the execution.

Moreover, situational data should be elaborated and any relationships between such information properly represented and correlated according to well-defined ontological constructs. We expect that the bulk of this sort of continuous "knowledge analysis and elaboration" phase will be performed mainly out of the service agents that will access and use it. It would be advisable to have special "knowledge manager" agents injected in the environment, in charge of properly analyzing and correlating such diffused situational data. Distributing such analysis activity among different actors helps achieving better scalability and reduces the reasoning capability that a service agent should have (we do not want a heavyweight rational service agent).

The final conceptual outcome of the above knowledge organization and analysis phases is the formation of so-called *knowledge networks*, in which all information about individual contexts are properly represented, organized and correlated, and around which semantically-enriched stigmergic interactions among individual agents can take place. Distributing such knowledge in the environment and hierarchically among agent aggregations, service agents can self-organize their activities using "cognitive stigmergy" approaches [17]. As anticipated earlier, the distributed knowledge network is expected to play the part of a *high-level intelligent and dynamic environment*, useful in particular for those self-organizing services that use the environment as a mediator for their local stigmergic interactions. *Self-adaptation* and *self-organization* would be driven by more sophisticated application-level knowledge data, other than simple pheromones value to react, and this will enable more robust and adaptive configuration patterns (e.g. the knowledge network can be used to enforce a more semantic control over a set of swarm agents). In addition, scattering context information among aggregate agents allows to make services situation-aware with different degrees of granularity: locally relevant situational data are consumed in place, while components are allowed also to reason about more global situational data, interrogating the distributed dynamic knowledge network: service components can "navigate" through the available knowledge hierarchy to attain, on demand, the degree of contextual awareness they require.

IV. CONCLUSIONS

The continuous growth in ubiquitous and mobile network connectivity, together with the increasing number of networked computational devices populating our everyday environments, call for a deep rethinking of traditional communication and service architectures. In this paper, we have focused on communication services, and have analyzed the key characteristics and features that a proper innovative component model for the effective development and deployment of autonomic (i.e., self-organizing, self-adaptive, self-healing) communication services should exhibit.

The results of our analysis can be simply summarized as follows:

- Such new component model should be general-purpose, able to enforce autonomic behaviour in both the forms of self-adaptation and self-organization, able to handle "situatedness" in complex knowledge environments, and should tolerate scalable forms of dynamic aggregation.
- Multiagent systems researches can play a major role in the definition of such component model and, more in general, in the advance of the autonomic communication research area. Nevertheless, as this paper envisioned, their scope should be limited by a clear and suitable component model, tailored to the requirements of autonomic communications. Such a model is the aim of the CASCADAS project in the future years.

V. ACKNOWLEDGEMENTS

Work supported by the European Commission within the Integrated Project CASCADAS (IST-027807) "Component-ware for Autonomic, Situation-aware Communications, And Dynamically Adaptable Services" (FET Proactive Initiative, IST-2004-2.3.4 Situated and Autonomic Communications).

REFERENCES

[1] The Autonomic Communication Forum, www.autonomic-communication-forum.org.

[2] J. L. Arcos, M. Esteva, P. Noriega, J. A. Rodriguez and C. Sierra, "Engineering Open Environments with Electronic Institutions", in Journal on Engineering Applications of Artificial Intelligence, 18(2):191-204, 2005.

[3] I. Carreras, I. Chlamtac, H. Woesner, C. Király, "BIONETS: BIO-inspired NExt generaTion networks" in the Proceedings of the 1st IFIP Workshop on Autonomic Communications, LNCS No. 3457, Springer Verlag, 2004.

[4] S. Dill, R. Kumar, K. Mccurley, S. Rajagopalan, D. Sivakumar, A. Tomkins, "Self-Similarity in the web", ACM Tran14tions on Internet Technology 2(3):205-223, 2003.

[5] S. Dobson, "Putting meaning into the network: some semantic issues for the design of autonomic communications systems", in the Proceedings of the 1st IFIP Workshop on Autonomic Communications, LNCS No. 3457, Springer Verlag, 2004.

[6] D. Estrin, D. Culler, K. Pister, G. Sukjatme, "Connecting the Physical World with Pervasive Networks", IEEE Pervasive Computing, 1(1):59-69, Jan. 2002.

[7] K. Fisher, "Holonic multiagent systems - theory and applications", in the Proceedings of the 9th Portuguese Conference on Progress in Artificial Intelligence (EPIA-99), LNAI Volume 1695, Springer Verlag, 1999.

[8] A. Giret and V. Botti, "Holons and agents", in Journal of Intelligent Manufacturing, vol. 15, pp. 645-659, Kluwer Academic Publishers, 2004.

[9] J. Hightower, G. Borriello, "Location Systems for Ubiquitous Computing", IEEE Computer, 34(8):57-66, Aug. 2001.

[10] J. Kephart, D. M. Chess, "The Vision of Autonomic Computing", IEEE Computer, 36(1):41-50, January 2003.

[11] H. Liu, M. Parashar, "Component-based Programming Model for Autonomic Applications", in the Proceedings of the 1st International Conference on Autonomic Computing, New York, NY, USA, 2004.

[12] A. Manzalini, F. Zambonelli, “Towards Autonomic and Situation-Aware Communication Services: the CASCADAS Vision”, 1st IEEE Workshop on Distributed Intelligent Systems, Prague (CZ), June 2006.

[13] A. Pentland, “Socially Aware Computation and Communication”, IEEE Computer, 38(3):63-72, March 2005.

[14] The European Commission, “The Situated and Autonomic Communications Initiative”, http://cordis.europa.eu/ist/fet/comms.htm.

[15] T. Sandholm and V. Lesser, “Coalitions Among Computationally Bounded Agents”, in Artificial Intelligence, Special Issue on Economic Principles of Multi-Agent Systems, Vol. 94, N. 1, pp. 99-137, 1997.

[16] O. Shehory, S. K. Sycara and S. Jha, “Multi-agent Coordination through Coalition Formation”, in Lecture Notes in Artificial Intelligence, n. 1365, pp. 143-154, Springer Verlag, 1997.

[17] L. Tummolini, C. Castelfranchi, A. Ricci, M. Viroli, A. Omicini, “Exhibitionists and Voyeurs do it better: A Shared Environment Approach for Flexible Coordination with Tacit Messages", Environments for MultiAgent Systems. LNAI 3374, Springer-Verlag, January 2005.

[18] R. Want, “An Introduction to RFID Technology”, IEEE Pervasive Computing, 5(1):25-33, Jan.-March 2006.

[19] F. Zambonelli, “Self-management and the Many Facets of Nonself”, in IEEE Intelligent Systems, Vol. 21, N. 2, pp. 50-56, 2006.

A QoS-Aware Architecture for Multimedia Content Provisioning in a GRID Environment

Fabrizio Messina, Giovanni Novelli, Giuseppe Pappalardo, Corrado Santoro, Emiliano Tramontana
Dipartimento di Matematica e Informatica
Università di Catania, Viale A. Doria, 6 - 95125 Catania, Italy
{messina, novelli, pappalardo, tramontana}@dmi.unict.it, csanto@diit.unict.it

Abstract— **In this paper, we propose a mobile agent infrastructure to support "Quality of Service" (QoS) parameters in multimedia content streaming. The infrastructure is able to face the issues concerning multimedia content transformation, in order to ensure that the client-side QoS is met. A Grid computing platform is exploited and content adaptation is performed through appropriate agents that are allocated as jobs and executed on Grid hosts. A location service is therefore devised and made available within the Grid system, which helps finding the hosts whose geographic position and network connections minimise the delays required to move the desired multimedia content from the storage to the processing site, and (after suitable transcoding) from this to the requesting client.**

Keywords: mobile agents, agent-based applications, GRID, QoS, Globus.

I. Introduction

Nowadays a large number of multimedia coding formats exists. Each format not only needs a proper decoder/player at the client side but also a different transport protocol featuring timing characteristics that require a network offering mechanisms to guarantee throughput, latency, packet loss ratio, jitter, etc. [2], [14], [17]. Such "Quality of Service" (QoS) requirements [10], if not fulfilled, can cause delays or interruptions in content playing and thus result in a service whose quality could be lower than expected.

As for coding scheme, even if some well-known general-purpose players, like Windows^{TM} Media Player or Xine, are able to support the majority of multimedia formats, there are still some schemes (e.g. QuickTime^{TM} MOV or Realplayer^{TM} RAM) that require their own players. Moreover, some players are not able to adapt the content to the quality of the connection used by the client, which could be too slow for the selected multimedia file, thus preventing proper reproduction. As a result, sometimes a multimedia content cannot be retrieved, played and used at client side, because the player is not available, the protocol is not supported, the client connection has not enough bandwidth or client performance is inadequate.

The reason behind this is due to the fact that current multimedia provisioning solutions give the client the responsibility of adapting the received content to both client and network capabilities. This may well be deemed unreasonable if we think that, in general, servers have more computational power and performances than traditional desktops or laptops, so having them adapt the provided content to each client seems more appropriate. Moreover, when multimedia provisioning is not performed by single servers but a *Grid computing environment* is employed [5], server-side on-line adaptation of streaming becomes not only feasible but also preferable: thanks to the ability of managing and offering a huge amount of storage space and CPU power, a Grid can be used not only to store and provide multimedia contents, but also to perform multimedia transcoding in order to satisfy at best the QoS parameters requested by the user, taking also into account network capabilities.

By considering the issues and goals outlined, this paper proposes a software architecture, based on mobile agents running in a computational Grid, for multimedia content provisioning and QoS satisfaction[1]. The main aspect of the proposed solution is that it is able to share the content adaptation cost between the client and the server, thus overcoming the problems described. A location service is also included, which aims at finding the host whose geographic position and network connections capabilities are able to minimise the time required to move the desired multimedia content *(i)* from the storage to the processing site, and *(ii)*—after suitable transcoding—from this to the requesting client.

The paper begins (Section II) with a description of a use-case that serves as a reference scenario to introduce the basic model of the solution. Then Section III illustrates the software architecture of the solution as integrated in a Grid environment, thus explaining how the main services of a Grid can be exploited for our purpose. Section IV describes the prototype implementation of the architecture, which has been developed, using GridSim. Section V discusses and compares other approaches. Section VI concludes the paper.

II. Use-Case Scenario and System Model

We consider a scenario in which a user asks for the retrieval and playing of a multimedia content by means of a client program or a web interface. We suppose that the client/player, when it connects to the server, specifies some *Quality of Service (QoS)* parameters regarding its capabilities, such as encoding scheme it is able to support, the speed of the connection used, etc. In particular, two sets of QoS parameters

[1]This work has been (partially) carried out within the Web-Minds, QUASAR and PI2S2 projects, funded by the Italian Ministry of Education, University and Research (MIUR), and the TriGrid VL project, funded by Regione Sicilia.

Application QoS	Network QoS
Encoding Scheme	End-to-end Throughput
Compression Scheme	End-to-end Throughput Jitter
Compression Ratio	Propagation Delay
Sampling/Frame Rate	Propagation Jitter
Picture Size	Packet-loss ratio
Colors	
Channels	

TABLE I
APPLICATION AND NETWORK QOS PARAMETERS

can be considered, which are summarised in Table I and explained in the following:

- **Application QoS parameters.** They are those parameters concerning application-level multimedia handling and include *encoding scheme*, *compression scheme* and *compression ratio*, for all types of contents, *sampling rate*, *sampling size* and *number of channels* for audio contents, and *frame rate*, *picture size* and *number of colors* for video data.
- **Network QoS parameters.** They refer to the capability of the network and subnetworks located on the path from the client to the server. Such parameters include the *end-to-end throughput*, the *propagation delay*, the *propagation jitter*, etc.

When the server receives a client's request, it has to try to fulfill it by matching the parameters specified with those of the multimedia content to be retrieved, also considering the capabilities of the network that has to carry the streamed data: if they differ or the network is not able to deliver the content according to the requirements, an adaptation would be needed, the complexity and feasibility of which depend on how much requested and offered contents differ from each other, and the type of conversions they imply.

Following such a use-case scenario, our solution aims at proving an efficient and reliable way to perform content transcoding and adaptation in order to fulfill at best the requirements of a client program. The basic model of the solution, which is based on *mobile agent technology*, exploits two agents, called ClientProxy and ServerProxy, running on the client and the server respectively, which act as "proxies" intercepting and adapting multimedia data. Basically (see Figure 1), the ClientProxy is designed as able to talk with the client application/player; it knows client's abilities in handle multimedia data and thus its application QoS parameters. On the other hand, the ServerProxy is able to handle both client's application QoS and the format of the multimedia content requested. Both agents start at the client side and thus both knows how to support the player characteristics, but the ServerProxy is a mobile agent; when the content streaming has to begin, the ServerProxy migrates to the server and both agents start to collaborate as follows:

1) The ServerProxy agent retrieves multimedia content and analyses it.
2) Both agents, according to the application QoS parameters of both the client and the content, and knowing the capabilities of the transfer link (network QoS parameters), establish an *inter-agent protocol* in order to transfer the multimedia content. Therefore, the ServerProxy performs the appropriate conversions and QoS adaptions to allow data transmission through the network with the negotiated inter-agent protocol.
3) The ClientProxy agent receives the data using the inter-agent protocol and performs the final conversions in order to supply data to the player with application QoS parameters requested.

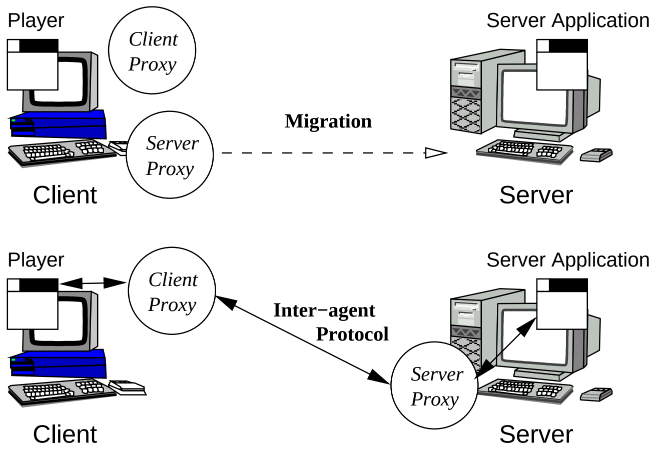


Fig. 1. Working Scheme of the Solution

Roughly speaking, this solution provides a significant advantage since it (basically) is able to avoid QoS loss. In fact, traditional solutions are based on some general purpose QoS translation algorithms running at server side, but such algorithms are not able to cover all the possible cases needed by a client. On the other hand, in the proposed solution, the transcoding part running in the server—the ServerProxy — originates from the client machine: it *exactly* knows what are the client's requirements and thus can encapsulate the right algorithms to perform a correct transcoding.

A significant drawback of the solution is however the computational power needed, at server side, to perform transcoding in the case of many concurrent requests. To face such an issue, we consider our solution as running on a Grid infrastructure since it can provide the power needed; to this aim a suitable architecture has been designed and detailed in the following Section.

III. RUNNING TRANSCODING AGENTS IN A GRID ENVIRONMENT

For obtaining the transcoding of a multimedia content, agents are allocated and executed in capable hosts of a Grid system. The most widespread software system used for Grid environment is the Globus Toolkit (GT) [13]. This includes services to access and monitor computing and data resources, enforce security, allow users to send and execute their applications. Computing resources have a specialised role according to their characteristics. Hence, a host providing a large amount of storage space is a *Storage Element (SE)*; a host holding a certain number of CPUs that can execute jobs is a *Computing Element (CE)*; whereas users are able to log into GT, send their commands and receive results, by means of a host called *User Interface (UI)* [6], [4].

In order to determine the host where the transcoding agent should run it is appropriate to take into account: (i) the location of the agent, (ii) the location of the user requesting the multimedia content, and (iii) the location of available hosts.

The services necessary for performing this selection are provided by the architecture components described in the following (see Figure 2).

In order to make the multimedia contents available to users, these are initially transferred into available SEs. Each content is characterised by its name, encoding, length, description, size, etc.

A user requests a multimedia content, by running a client software on his/her UI. The request is characterised by: (i) the name of the multimedia content or its description, and (ii) the needed encoding format.

The request is processed by service *MuM* (for *Mu*ltimedia *M*ining) that finds available versions of the content. When transcoding from an available format to the requested version is required, other services are involved in order to: find an appropriate CE that will host one or more capable agents, find and transfer both the source version of the content and the transcoding agent into the selected CE. The output of the transcoding agent, which is a new format for the initial multimedia file, is then provided to the user on his/her UI and also stored to some SE so as to better serve further requests for the same file and for the newly available format.

A. Services for Selecting Hosts

The service *MuM* is able to find the locations of a user requested multimedia content (see in Fig. 2 interactions (1, 2, 3)). *MuM* returns the location, in terms of SE address, where each format of the content can be found. *MuM* is organised as a repository based on the existing Grid Resource Information Service (GRIS) [3][2]. *MuM*'s data are updated when new contents are inserted, new formats are available, and existing ones are re-organised (e.g. moved to better support accessing them).

Service *Agent Locator (AL)* is responsible to store the location of transcoding agents. While agents are initially stored only on the users' hosts, they are re-located to several CEs according to the requests for transcoding the multimedia contents. To minimise the re-location overhead, agents are kept on the CEs even when idle, i.e. after contents processing has finished. Service *AL* traces where each agent is, thus when *AL* is queried for a given transcoding agent (see (4) in Fig. 2), it returns the list of CEs addresses where that agent is located.

Service *CE Locator (CEL)* traces the availability and CPU capacity of CEs. This service initially asks known GRIS about data concerning the number of hosts in a CE, and the characteristics of the hosts, in terms of CPU type and amount of RAM. Additionally, *CEL* periodically asks the length of the job queue on a CE. All these data are asked asynchronously from a user query and stored locally by *CEL*. This helps reducing the performance penalties given by the interactions with GRIS. When an agent has to be allocated, *CEL* is asked for available hosts (see (5) in Fig. 2) and returns a list of CE addresses, whose hosts are idle or lightly loaded.

[2]GRIS is part of the Monitoring and Discovery Service (MDS) provided by the Globus Toolkit.

Finally, service *Resource Finder (ReF)* selects a CE for hosting transcoding agents. In selecting a CE, *ReF* tries to minimise latencies due to the network and achieve the shortest response time for having the adapted content. Latencies are calculated according to the following three "distances": (i) the "distance" $d(SE_i, CE_j)$ between the SEs where the needed content is stored (provided by MuM) and the available CEs (provided by CEL); (ii) the "distance" $d(CE_j, UI)$ between available CEs and UI; and (iii) the "distance" $d(CE_j, A(H_k))$ between the available CEs and agent A that can be found in one of several hosts H_k (the locations where agent A is found is provided by AL).

Each "distance" is measured as the number of seconds necessary to deliver 100Mb across two end points, thus it takes into account both the available bandwidth and the latency, i.e. the physical space that has to be crossed. Measures between each two pairs of known sites are performed off-line and stored.

The selected sites $SE_i^{'}$, $CE_j^{'}$, representing, respectively, the host from which the source content will be loaded and the host allocating the transcoding agent are those that minimise

$$d(SE_i, CE_j) + d(CE_j, UI)$$

for i, j ranging over the available sets of SE_i, CE_j.

Once $CE_j^{'}$ has been determined, the value $d(CE_j^{'}, A(H_k))$ is minimised, i.e. the host $H_k^{'}$ is chosen from which the agent will be transferred.

Note that *ReF* collects the necessary data and determines the CE that should host the transcoding agents, the CE providing such an agent, and the SE where the source multimedia content is stored. In Fig. 2, they are CE_1, CE_2 and SE_2, respectively.

In order to activate multimedia content transformation and delivery, *ReF* submits a job to the selected CE, e.g. CE_1, (see (6) in Fig. 2). This job executes the following set of steps. Firstly, it collects the necessary parts, i.e. the transcoding agent from CE_2 (6.1) and the multimedia content from SE_2 (6.2). Secondly, it executes the agent that will process the content. Finally, the result of content transformation is directly passed on to the user front-end.

B. Handling QoS for Multimedia Contents

In order to have a short response time between a user request for a multimedia content and the transcoded version, the capabilities of the supporting infrastructure, i.e. available bandwidth and computing "power" have to be determined beforehand.

The preferences set by a user in terms of frame size and coding format for a multimedia content affect the activities that are performed by the infrastructure to serve the user request.

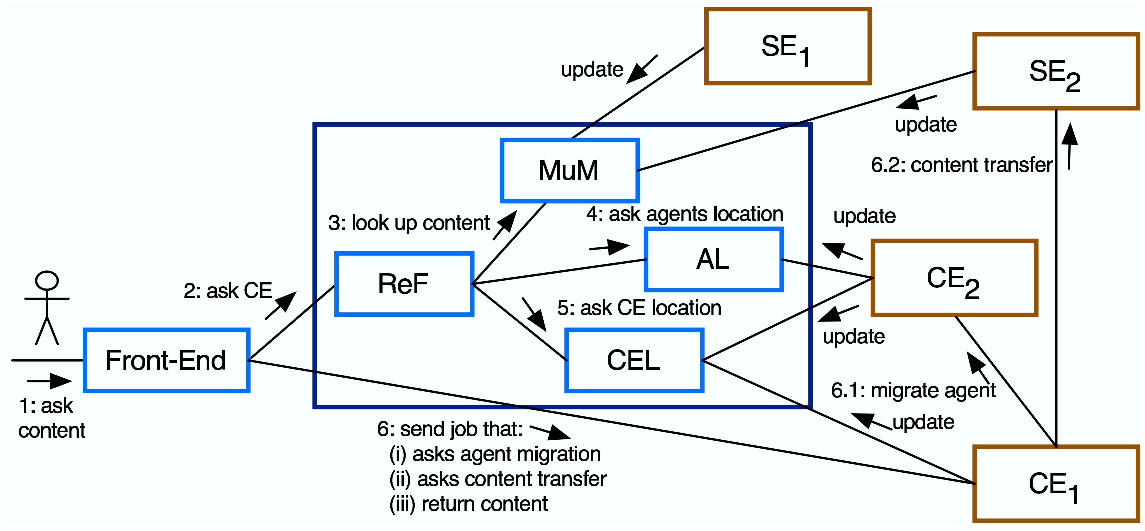


Fig. 2. Overview of interactions between front-end and services.

One of these activities consists in the multimedia processing, another one is the a priori reservation of bandwidth and hosts.

At least one capable host will be reserved to run the transcoding agent. However, just one host could be not sufficient for the necessary processing. When the estimation of the time needed to sequentially transcode the content on a single host would result in an output rate that is less than the rate necessary to ensure that the content can be seen/listened smoothly, then a parallel transcoding activity is organised. For this parallel processing, the initial multimedia content is fragmented and each chunk processed by an agent on a dedicated host.

Let us first express the way we estimate the time needed to transcode a content from a format to another.

We have processed several videos having different resolution (see Table II and III) and observed that processing time is mainly related with the frame resolution and size in bytes of the frames to be processed. I.e. for the same video, having e.g. a resolution of 320x240 pixels, the processing time is almost proportional to the size (in bytes) of the JPEG frame that has to be transcoded.

TABLE II
CHARACTERISTICS OF SAME SAMPLE VIDEOS.

content	source	resolution	length (s)	size (MB)
chinese	AVI	320x191	152	20.3
wr	AVI	320x240	14	1.0
cartoon	AVI	480x272	115	7.6
dhl	AVI	640x480	43	17.7

In Table III, we report for several videos (whose characteristics are found in Table II) the time in milliseconds necessary to convert from AVI to MPEG, MJPEG, h263p and rv10 (see the 4 right-most columns, respectively) several chunks of the content lasting 2 seconds each and having 50 or 60 frames, as indicated in the column frames. Transcoding is performed from the set of frames (each stored as a JPEG file) and whose overall size in megabytes is indicated in column size.

TABLE III
TRANSCODING TIME FOR SOME SAMPLE VIDEOS.

content	frames	size	MPEG4	MJPEG	h263p	rv10
chinese	1– 50	0.27	87	79	107	110
	51–100	1.51	181	156	179	181
	101–150	1.25	182	139	180	183
	151–200	1.35	191	148	189	188
wr	1– 50	2.28	247	212	244	238
	51–100	1.69	204	180	200	206
	101–150	1.76	204	186	205	202
	151–200	1.55	193	170	193	194
cartoon	1– 60	1.76	240	240	230	250
	61–120	1.64	280	260	300	290
	121–180	3.32	380	340	370	380
	181–240	4.10	470	400	450	450
dhl	1– 60	8.00	935	894	901	891
	61– 120	6.77	853	738	835	842
	121– 180	6.64	808	733	815	806
	181– 240	8.95	956	905	953	923

The reported values are just an excerpt of the experiments we have performed. In other experiments we have converted the whole reported video, moreover other videos have been processed in a similar way.

The estimation function for the time needed to transcode into MPEG4, but the same applies to the other output formats, is the trend function calculated according to the values stored in a database of observed times. The resulting trend function takes as input the resolution and frame size as parameters and returns the estimated time.

In order to calculate the number of hosts needed for the transcoding, we compare the estimated transformation time and the length in seconds of the processed video. When transformation time is greater than processed video then we must use more than one host for the transformation and the number of needed hosts is the minimum natural number bigger than the ratio between transformation time and length.

Bandwidth reservation aims at readily transfer data from a disk to the selected CE that will perform transcoding and from the CE to the user device.

The amount of bandwidth necessary is calculated as the ratio between size of the content and length (this is just an approximation since the amount of bytes per frame are not constant for the whole video).

IV. Simulating Environment

In order to assess the timing characteristics of the proposed software infrastructure and how it reacts when a large number of users send requests for accessing multimedia contents, we are in the process of developing a model of the proposed infrastructure using the simulation environment GridSim [8]. GridSim is a Java toolkit that can be programmed to simulate a Grid system, thus it offers support, in terms of classes, for simulating CEs, SEs, Virtual Organisations, etc.

In our proposed infrastructure, transcoding agents can move and communicate within a Grid environment. The underlying support for allowing such agents to move, communicate, etc. is an agent-platform, such as Jade [1]. For simulation purposes, on top of GridSim we have developed a class library that models the activities of an agent platform (similar to Jade). Such a library, called *GAP (Grid Agent Platform)*, includes the classes: AgentPlatform, AgentSite, and Agent.

Class AgentPlatform represents a federation of sites offering facilities that allow agents to move, find each other, communicate, etc. This class allows the access to services that are expected from an agent platform, such as: (i) a registry informing about available agents on any site, simulated as class DirectoryFacilitator; and (ii) an observer for all the movements of agents, simulated as class NetworkMonitor.

Class AgentSite is responsible to hold data about the location and the state (e.g. running, stopped, idle) of existing agents in a given CE within the Grid system. AgentSite extends GridSim class GridSim and holds an instances of GridSim class GridResource, which represents the CE.

Class Agent represents an agent and holds data about the identifier, state, type, and size of the agent. Each Agent instance simulates the computation performed by an agent, which can be affected by events that are notified to it through messages. The events that can affect an agents are: stop, resume, terminate, and migrate. Messages are exchanged between agents while performing their computation. Each time an agent migrates or change its state, both classes DirectoryFacilitator and AgentSite are informed. An Agent instance simulates some processing by generating jobs and submitting them into the CE where the agent is located. A generated job is an instance of GridSim class Gridlet.

The network connections between CEs and SEs are modelled according to GridSim class Link. Instances of such a class are set with values representing bandwidth and delay of the network link, as well as the instances of GridResource it connects.

An additional User class models the user requests, thus the ability to ask for a content and receive a result.

Along the above classes, all the services of the proposed infrastructure (i.e. ReF, MuM, AL and CEL) are modelled as classes that execute on the Grid system, process requests and provide corresponding results.

For running a simulation, the number of CEs, transcoding agents types and users are set. The number of instances of AgentSite created is according to the the given number of CEs chosen. Each AgentSite instance is set with a CE identifier. Each type of transcoding agent is modelled as an instance of Agent. Several instances of Agent are created and each is set with: a value for the size of the code of the agent (in bytes); its initial location, using an instance of AgentSite; and the state of the agent, as idle. Finally, instances of User are created.

Once a simulation is executed, GridSim output is a report that for each modelled class describes the number of received requests, the amount of time necessary to process the requests, the time spent waiting for replies, etc. This report will provide valuable feedback on all the assumptions concerning the expected time of network connections to transfer multimedia contents, hosts to process fragments of contents, infrastructure to select hosts for processing, etc.

V. Related work

Several existing approaches, including [7], [15], generate off-line different transcoded versions of a multimedia content, i.e. before users request a format this is prepared and stored into the disk. Thus serving a request is just a matter of reading the content from the disk. Of course, generating different formats for a given content requires a large amount of computing resources, thus these approaches recur to parallel processing, using clusters of hosts, for the initial content.

An approach for the on-line transcoding of multimedia contents by means of software system for a Grid environment is proposed in [16]. In such an approach, a broker component finds and dynamically allocates transcoding jobs on available Grid resources, according to the incoming requests. Similarly to our approach, the broker component selects capable and unloaded host for ensuring a short response time. However, we additionally take into account both static and dynamic network conditions. Moreover, in this approach the server hosts have to be equipped with all the necessary transcoding libraries beforehand. In contrast, our approach employs agents that move to available hosts, thus we do not need to install transcoding support into all the Grid hosts.

On-line transcoding is also faced in [11], where the authors propose a support for fitting multimedia content into mobile devices connected to a wireless network. Transcoding, which is based on changing frame size, color depth and Q-scale, is performed inside a HTTP proxy, therefore the approach is suitable to adaptation of video content that could be found on the web. In contrast to our proposal, this approach considers only some parameters that are fixed at server-side and does not perform automatic parameter adaptation on the basis of network connection monitoring. Moreover, in our approach agents are dispatched from the client, they are aware of the target player's capabilities and are thus able to adapt the content at best, taking also into account the varying communication conditions.

When having to serve requests for contents, *Content Delivery Network* (CDN) [9] can bring benefits in terms of efficient delivering, since contents are replicated and relocated by the CDN according to request types and origins. Our approach is aimed not only at finding the nearest location for the requested multimedia content, but especially for on-the-fly generation of new a content format through transcoding according to client requirements.

Adaptive Web Systems (AWS) [12] provide unaware users with customised versions of contents, selected at server side on the basis of user profile, location, access mode, terminal, etc. In contrast, our approach suggest that clients have an active role in the content adaptation process, i.e. they are represented by their agents for this purpose. In principle, this permits more sophisticated and dynamic forms of adaptation, but of course requires a supporting infrastructure and a more capable and aware client.

VI. Conclusions

This paper has presented a mobile agent architecture for multimedia content provisioning and QoS adaptation which exploits a computational Grid, for its CPU power and storage space availability. The basic working scheme of our proposal entails two mobile agents, one at client side and the other at server side, what cooperate with each other to perform multimedia adaptation and transcoding, thus aiming at adapting the QoS of the content to the QoS requested by multimedia player.

The architecture consists of a set of components entailed with the task of finding *(i)* the storage element that possesses the multimedia content to be played and *(ii)* the computing element where to host the mobile agent in charge of server-side transcoding. To this aim, some metrics are introduced, intended to minimize the distances between user and the computing element, and between the latter and the storage element where the multimedia content is stored.

In order to evaluate the feasibility of our solution, numerous tests have been performed to determine the time taken by various types of transcoding. A prototype implementation, running with a simulation tools, has also been developed, in order to evaluate not only the feasibility of the solution but also its performances.

As a future work, our goal is to package the system as a Globus Toolkit extension, so as to obtain a complete and standard solution.

References

[1] F. Bellifemine, A. Poggi, and G. Rimassa. Developing multi-agent systems with a FIPA-compliant agent framework. *Software - Practice and Experience*, 31(2):103–128, 2001.

[2] Anjali Bhargava and Bharat Bhargava. Measurements and Quality of Service Issues in Electronic Commerce Software. In *Proceedings of IEEE International Conference on application-specific Software Engineering and Technology (ASSET-99)*, Texas, March 24-27 1999.

[3] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid information services for distributed resource sharing. In *Proceedings of the 10th IEEE Symposium On High Performance Distributed Computing*, 2001.

[4] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. In *Open Grid Service Infrastructure WG, Global Grid Forum*, June 22 2002.

[5] Ian Foster and Carl Kesselman, editors. *The Grid (2nd Edition): Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 2004.

[6] Fabrizio Gagliardi, Bob Jones, Mario Reale, and Stephen Burke. European DataGrid Project: Experiences of Deploying a Large Scale Testbed for E-science Applications. *Lecture Notes in Computer Science*, 2459, 2002.

[7] F. Gonzalez-Castano, R. Asorey-Cacheda, R. Martinez-Alvarez, E. Comesana-Seijo, and J. Vales-Alonso. Dvd transcoding with linux metacomputing, 2003.

[8] Grid Computing and Distributed Systems (GRIDS) Laboratory. *GridSim: A Grid Simulation Toolkit for Resource Modelling and Application Scheduling for Parallel and Distributed Computing*, 2005. http://www.gridbus.org/gridsim/ .

[9] Markus Hofmann and Leland R. Beaumont. *Content Networking*. Morgan Kaufmann, 2005.

[10] ITU-T (CCITT). Recommendation. Terms and Definitions Related to The Quality of Service and Network Performance including Dependability, August 1994.

[11] Ricardo Oliveira Parixit Aghera, Advait Dixit and Vidyut Samanta. Wireless middleware: Dynamic video transcoding. In *Communication Systems Software and Middleware*, New Delhi, India, jan 2006.

[12] Mike Perkowitz and Oren Etzioni. Adaptive web sites. *Communication ACM*, 43(8):152–158, 2000.

[13] The Globus Alliance. Home page and publications. http://www.globus.org .

[14] Andreas Vogel, Brigitte Kerhervè, Gregor von Bochmann, and Jan Gecsei. Distributed Multimedia and QoS: A Survey. *IEEE Multimedia*, 2(1):10–19, 1995.

[15] Gord Watts and Steve Forde. *Grid Computing: A Practical Guide to Technology and Applications*, chapter Grid Enabling Software Applications, pages 252–263. Charles River Media, Hingham, MA, 2003.

[16] Angelo Zaia, Dario Bruneo, and Antonio Puliafito. A scalable grid-based multimedia server. In *Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE)*, pages 337–342, 2004.

[17] John A. Zinky, David E. Bakken, and Richard D. Schantz. Architectural Support for Quality of Service for CORBA Objects. *Theory and Practice of Object Systems*, January 1997.

A Heterogeneous Multi-Agent System for Adaptive Web Applications

Andrea Bonomi, Giuseppe Vizzari
Department of Informatics, Systems and Communication
University of Milan–Bicocca
Via Bicocca degli Arcimboldi 8, 20126 Milano, Italy
{andrea.bonomi, vizzari}@disco.unimib.it

Marcello Sarini
Department of Psychology
University of Milan–Bicocca
Piazza dell'Ateneo Nuovo 1, 20126 Milan - Italy
sarini@disco.unimib.it

Abstract— **A web site presents an intrinsic graph–like spatial structure composed of pages connected by hyperlinks. This structure may represent an environment in which agents related to visitors of the web site are positioned and moved in order to track their navigation. To consider this structure and to keep track of these movements allows the monitoring of the site and of its visitors, in order to support the enhancement of the site itself through forms of adaptivity, carried out by specific interface agents. This paper presents a heterogeneous multi-agent system supporting the collection of information related to user's behaviour in a web site by specific situated reactive agents. The acquired information is then exploited by an application supporting the proposal of hyperlinks based on the history of user's movement in the web site environment.**

I. Introduction

A web site presents an intrinsic graph–like spatial structure composed of pages connected by hyperlinks. However, this structure is generally not considered by web servers, which essentially act as a sort of extended and specific File Transfer Protocol servers [1], receiving requests for specific contents and supplying the related data. Several web–based applications instead exploit the structure of the sites itself to support users in their navigation, generating awareness of their position. For instance, many e–commerce sites emphasize the hierarchical structure linking pages related to categories (and possibly subcategories), included products and their specific views, and remind users' relative position (i.e. links to higher level nodes in the tree structure). Some specific web–based applications, mainly bulletin boards and forums (see, e.g., phpBB[1]), are also able to inform users about the presence of other visitors of the web site or even, more precisely, of the specific area of the site that they are currently viewing. Web site structure and users' context represent thus pieces of information that can be exploited to supply visitors a more effective presentation of site contents.

Different visitors, however, may have very different goals and needs, especially with reference to large web sites made up of several categories and subcategories. This consideration is the main motivation for the research in the area of adaptive web sites [2]. The various forms of adaptation may provide a customization of site's presentation for an individual user or even an optimization of the site for all users. There are various approaches supporting these adaptation activities, but they are generally based on the analysis of log files which store low–level requests to the web server: this kind of file is generally made up of entries including the address of the machine that originated the request, the indication of the time and the resource associated to the request. In order to obtain meaningful information on users' activities these raw data must be processed (see, e.g., [3]), for instance in order to collapse requests related to various elements of a single web page (e.g. composing frames and images) into a single entry. Moreover, this kind of information must be further processed to detect groups of requests that indicate the path (web pages connected by hyperlinks) that a user followed in the navigation. Recent results [4] show that this kind of analysis, also referred to as web usage mining, could benefit from the consideration of site contents and structure.

This paper proposes to exploit the graph-like structure of a web site as a Multi–Agent System (MAS) environment [5] on which agents representing visitors of the web site (hereafter *user agents*) are positioned and moved according to their navigation. In particular, in this case, the environment is a virtual structure which allows the gathering of information on user's activities in a more structured way, simplifying subsequent phases of analysis and adaptation of site contents. Furthermore, part of the adaptivity could be carried out without the need of an off-line analysis, but could be the result of a more dynamic monitoring of users' activities. In particular, the paths that are followed by users are often related to recurrent patterns of navigation which may indicate that the user could benefit from the proposal of additional links providing shortcuts to the terminal web pages, as a sort of suggestion to the web site visitor. Index pages may thus be enhanced by the inclusion of links representing shortcuts to the typical destinations of the user in the navigation of the web site. Moreover, links between terminal content pages that are not provided by the static structure of the site can also be identified and exploited. Users without a relevant history (and also anonymous or unrecognized ones) may instead exploit the paths that are most commonly followed by site visitors. Moreover such an information could also be communicated to the webmaster suggesting possible modifications to the

[1]http://www.phpbb.com/

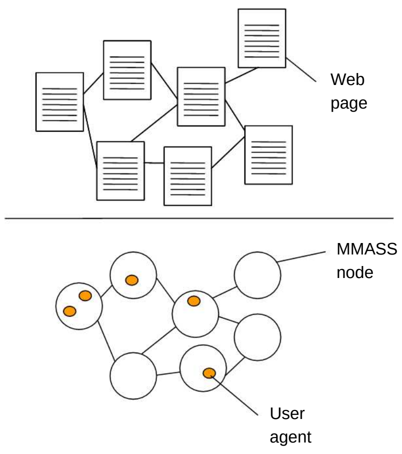


Fig. 1. The diagram shows a mapping between a web site structure and an agent environment.

static predefined structure of the site. This approach provides thus both a support for site optimization, but also for the customization to specific visitor's needs and preferences.

The metaphor of a web site as an environment on which users move in search for information is not new (see, e.g., [6] but also more recent approaches such as [7]), and its application to web site adaptation resembles the emergent, collective phenomenon of trail formation [8] which can be identified in several biological systems. However, this proposal provides more than just gather information on users' behaviours for sake of web pages adaptation or navigation support, but exploits the MAS environment to provide users a means for mutual perception and interaction. In fact information related to users' positions on the environment representing the web site can also be used to supply them awareness information on other visitors which are currently browsing the same page or area of the site. Moreover, to keep track of this information allows the conception of a form of interaction among users that is based on their positions on the site. Essentially, more than just showing a user the other registered visitors that are "nearby" (i.e. viewing the same page or adjacent ones), the system could also allow to communicate with them. This form of interaction, in addition to the web page adaptation function, requires the adoption of a supporting technology that goes beyond the request/response model.

The overall system architecture requires thus proper *interface agents*, able to interact with user agents situated in the previously introduced environment in order to exploit the acquired information on users' behaviours. This second type of agent is totally different from user agents, both from a modelling point of view and with reference to the supporting technology. In fact the web interface agent must be active as long as the related web page is being viewed by a visitor and it must be able, in collaboration with the rest of the system, to proactively modify the page to improve the user's browsing experience. The overall system architecture includes thus heterogeneous agents collaborating to achieve this goal.

The following section describes the general framework of this approach, the mapping between the web site structure and agents' environment, while Section III introduces the gathered information on agents' movement in their environment. Section IV describes an application providing the exploitation of this information for the adaptation of web pages, both for customization and optimization. The adopted technology supporting the design and development of the related interface agent is introduced, and discussed with reference to existing alternatives. A brief comparison of this approach and related work can be found in Section V, and finally concluding remarks and future developments will end the paper.

II. Site Structure and Reactive User Agents

A web site is made up of a set of HTML pages (generally including multimedia contents) connected by means of hyperlinks. It is possible to obtain a graph-like structure mapping pages to nodes and hyperlinks to edges interconnecting these nodes. This kind of spatial structure could be exploited as an *environment* on which user agents related to site visitors are placed and move according to the related users' activities. A diagram showing a sample mapping among a web site and this kind of structure is shown in Figure 1.

This structure can be either static or dynamic: for instance it could vary according to specific rules and information stored in a database (i.e. database driven web sites). However, this kind of structure (both for static and dynamic web sites) can generally be obtained by means of a crawler (see, e.g., Sphinx [9] and the related WebSphinx project[2]); then it could be maintained by having periodic updates.

Given this spatial structure, a multi-agent model allowing an explicit representation of this aspect of agents' environment is needed to represent and exploit this kind of information. Environments for Multi Agent Systems [10] and situated agents represent promising topics in the context of MAS research, aimed at providing first class abstractions for agents environment (which can be more than just a message transport system), towards a clearer and more concrete definition of concepts such as *locality* and *perception*. There are not many models for situated agents, which provide an explicit representation of agent's environment. Some of them are mainly focused on providing mechanisms for coordinating situated agent's actions [11], other provide the interaction among agents through a modification of the shared environment (see, e.g., [12], [13]). An interesting approach that we adopted for this work is represented by the Multilayered Multi Agent Situated Systems (MMASS) [14] model. MMASS allows the explicit representation of agents' environment through a set of interconnected layers whose structure is an undirected graph of nodes (also referred to as sites in the model terminology; from now on we will use the term node to avoid confusion with web sites). The model was adopted given the similarity among the defined spatial structure of the environment and the structure underlying a web site. Moreover, the model defines a set of allowed actions for agents' behavioural specification

[2]http://www-2.cs.cmu.edu/ rcm/websphinx/

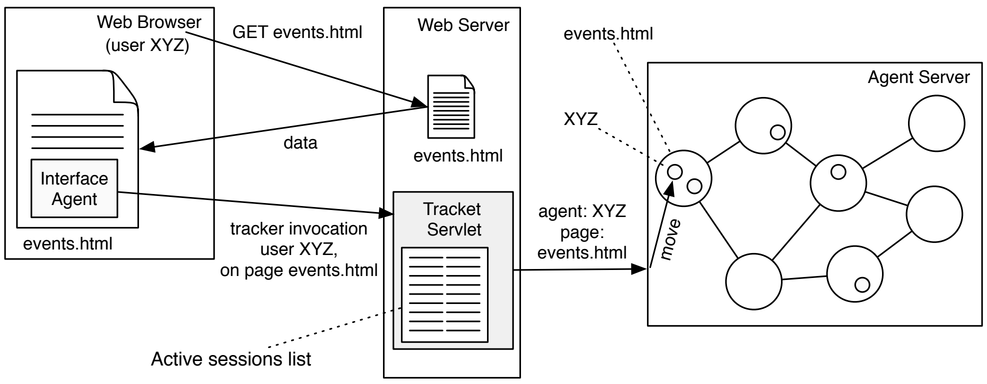


Fig. 2. A diagram showing how user actions influence the related reactive user agent through the capture of requests by the Tracker module.

(including a primitive for agents' movement); for this specific application, however, the constraint which limits the number of agents positioned in a node was relaxed. In fact there is no limit to the number of users that are viewing the same web page.

Moreover a platform for the specification and execution of simulations based on the MMASS model [15] was exploited to implement the part of the system devoted to the management of agents in their environments. The definition of spatial structure of the environment was supplied by the previously introduced crawler, while agents' movement is guided by external inputs generated by the requests issued by the related web site visitor. The general architecture of the system is shown in Figure 2: the *Agent server* module is implemented through the MMASS platform, while the *Web server* is a Tomcat servlet container hosting SnipSnap[3], a Java-based weblog and wiki software. The highlighted *Tracker* module is a implemented through a Java Servlet, which is invoked by every page of the site but does not produce a visible effect on the related web page. The Tracker is responsible for triggering the creation and the movement of agents related to visitors in the environment related to the web site structure. In particular, when a user makes his/her first page request the Tracker is invoked by the interface agent associated to the page. Then the Tracker tries to set a cookie on the client including the session information. If the cookie is accepted, it is possible to use the session information to identify the user; on the other hand, requests from clients not accepting cookies will not be monitored.

The management of agents creation and movement is not as simple as its intuitive description might indicate. In fact, the same user could be using different browser pages or tabs to simultaneously view distinct pages of the site. In other words, a user might be simultaneously following different trajectories in his/her web site navigation. In order to manage these situations, a user can be related to different agents, and his/her requests must be associated to the correct agent (possibly a new one). Finally, agents related to finished (or interrupted) user navigation should be eliminated by the system, storing the relevant part of their state in a persistent way, until the related user requires again a page of the site. In particular, remote users' requests may be divided into two main classes, according to their effects on the Tracker and Agent server:

- *creating a new agent*: whenever a new user requires a web page, the Tracker will invoke the Agent Server requiring the creation of an agent whose starting position is the node related to the required page; the same effect is generated by a request coming from an already registered user which was not present in the system, but in this case information related to previous user agents must be retrieved in order to determine the new agent's state; finally, when an already registered and active user requires a page that is not adjacent to its current one, a new agent related to the new browsing activity must also be created;
- *generating the movement of an agent*: when the viewer of a page follows one of the provided links, the related web browser will generate a request for a page that is adjacent to one of the related agents which must be moved to the node related to the required page; whenever there are two or more agents in positions that are adjacent to the required page, in order to solve the ambiguity and choose the agent to be moved, the Tracker will invoke the Session object in which it stores the current URL related to the viewed page.

The following section will describe how the raw information that can be gathered thanks to the above described framework can be processed in order to obtain higher level indications on users' behaviours. Since the interface agent collaborates to the user monitoring process, more details on this topic will instead be given in Section IV-B.

III. Gathered Information: Browsing Traces

This system allows to gather and exploit two kinds of information: first of all situated agents related to web site visitors have a perception of their local context, both in terms of relative position, adjacent nodes and presence of other visitors; second, agents may gather information related to the paths defined by the browsing activities or the related user in the site itself.

[3]http://snipsnap.org

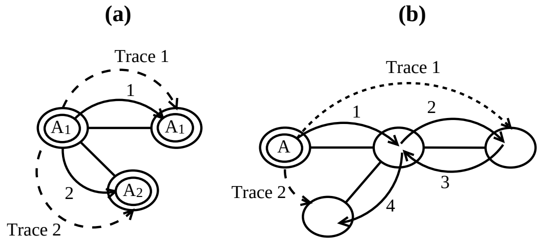


Fig. 3. A diagram describing two traces that are derived by a sequence of user requests.

There are inherent issues in determining in a precise way the actual users' activities on the web site, due to the underlying request/response model: the only available indications on these activities can be obtained by requests captured by the Tracker. In particular, we have an indication of the page that was required by a user and the time-stamp of the request. Starting from this raw information the system can try to detect *emerging links*, which are hyperlinks that are not provided by the structure of the site but can be derived by the behaviour of specific visitors. To this purpose, the concept of *trace* was introduced as a higher level information describing the behaviour of a user. A trace synthesizes a path followed by a user, from the web page representing his/her entry point, to a different point of the environment (i.e. another web page) which may represent an interesting destination. Every agent related to a visiting user is associated to a *temporary* trace, and it may generate several actual traces (also called *closed* traces) in the course of its movement in the environment.

Formally a trace is a three-tuple $\langle A_{Id}, Start, Dest \rangle$, where A_{Id} represents the identifier of the agent to which the trace is related, while $Start$ and $Dest$ indicate the starting and destination node related to the browsing sequence which generated the trace. A new trace is generated when a user enters the site, triggering the creation of a related agent. The starting trace has a null value for the destination node. Subsequent requests by the user generated following hyperlinks will bring the related agent to an adjacent node, and the the $Dest$ field of the corresponding trace will be modified in order to reflect user's current position. Non trivial traces provide $Start$ and $Dest$ nodes that are not directly connected by means of a hyperlink.

There are two relevant exceptions to the basic rule for trace update, that are related respectively to the *duplication* of a trace and to its *closing*. According to the previously introduced informal definition, a trace should be coherent in time and space. In fact, whenever the same user requires simultaneously two or more different pages he/she is probably following distinct search trajectories, possibly even related to different goals. In this case, as previously introduced, the Tracker will detect this situation and create additional agents that refer to the same user. Figure 3 shows two sample situations providing respectively trace duplication and closing: in (a) the user has chosen to open a hyperlink in a new browser page (request 1) and then has followed another link in the first browser page (request 2). According to the previously described Tracker behaviour, two agents are now associated to the user, and they are associated to different traces sharing the $Start$ field.

In (b), instead, the user has followed links 1 and 2 from the starting page, then he/she made a step back (request 3) and eventually moved to the last known position (request 4). The step back causes the closure of the temporary trace associated to the agent (Trace 1 in the Figure), and the creation of a new temporary one with the same $Start$ field (Trace 2). In this case the step back may have different interpretations: it could refer to a negative evaluation of the page contents but it could also indicate the fact that the user has found what he/she was searching for. An information that could be exploited to determine if the $Dest$ field of the trace was interesting for the user is the time interval between request 2 and 3: for instance, given Δt_d a threshold indicating the minimum time required to reasonably inspect the content of a specific web page, if $timestamp(3) - timestamp(2) < \Delta t_d$ then Trace 1 could be ignored. However, the mere interval between the two requests is not a safe indicator of the fact that the page was actually viewed and considered interesting.

In fact, the time spent on a web page is also important in order to determine when a temporary trace must be closed. In fact, whenever a user does not issue requests for a certain time we could consider that his/her browsing activity has stopped, possibly because he/she is reading the page related to the $Dest$ field of the trace associated to the related agent. In other words, every agent has a timer, set to the previously introduced threshold Δt_d, which is set when the agent is created and it is reset whenever it moves. The action associated to this timer specifies that its temporary trace becomes closed, and a new timer is set: the action associated to this second timer caused the disappearance of the agent from the system, and the storage of the related state.

It is important to note that even anonymous visitors (i.e. non authenticated ones) whose clients are accepting cookies, can be tracked and can thus generate traces, although anonymous ones. The latter can be exploited for sake of web optimization but are not relevant for sake of user specific site customization.

User agents provide thus a support to interface agents by monitoring users' behaviours and, in this specific case, selecting relevant traces. Figure 4 shows how the user agents interact with the interface agents to provide them with relevant information for page adaptation, but more details on this topic will be provided by the following section.

IV. The Web Interface Agent

The aim of the Interface Agent is to improve the browsing experience of a user by adapting the page he/she is currently viewing to his/her preferences, needs or habits. To do so, it must be active during the time–span in which the page is visualized by the browser, and it must be able to dynamically alter its appearance. To do so, it must also be able to interact with the previously introduced system to be informed about past user's behaviour. In other words the interface agent is

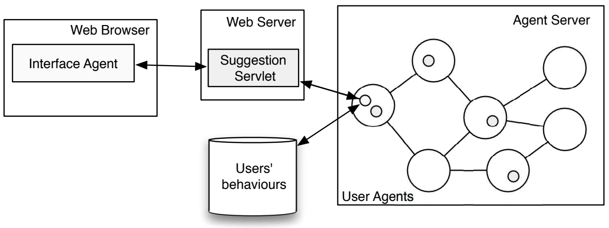


Fig. 4. A diagram showing the interaction among an interface agent, the user agent (in the MMASS environment) and the users' behaviors database.

a client–side component, "living" in the web browser and interacting with it in a proactive way, as shown in Figure 4.

In the following sections, we describe the technology adopted to implement the interface agent, comparing it with other currently available technologies that could have been selected to develop this kind of client-side web application. Then the behaviour of the interface agent is briefly introduced, focusing on its setting in the overall architecture and on the adopted strategy for page adaptation.

A. Technologies for Web Interface Agents: Java Applet, Flash and AJAX

Today there are several technologies suitable to develop rich client–side web applications, and in particular interface agents able to "live" in a common web browser. The most common are Java Applet, Macromedia (now Adobe) Flash and AJAX. We intentionally chose not to consider recent browser extensions and plug-ins for the visualization of 3D virtual environments, and to focus on more traditional forms of web browser interfaces.

Java Applet[4] is the oldest technology used to provide interactive features to web applications. An applet is a Java software component that runs in a Web browser using the Java Virtual Machine. Applets can be included in HTML (or XHTML) pages in the same way as an image or another multimedia content, and they are executed in a *sandbox*, an infrastructure preventing them from accessing client's local data (though there may be exceptions to this principle, and in particular *trusted applets*). This kind of approach is very powerful because applets can exploit all the Java API: they can, for example, generate complex user interfaces, with a rich multimedia support (e.g. 3D graphics, sound, movies), or they can interact with server–side application via Web Services, Java RMI (Remote Method Invocation) or CORBA. It is possible to develop very complex applications using common Open Source Java IDEs (like Eclipse or Netbeans) and run them in web browser as applets. Though Java Applet can be a suitable technology for many complex web application, it is difficult to implement an interface agent with an applet because of its lack of integration with the web browser. An applet is in fact confined in a sandbox and cannot manipulate the data of the page in which it is being executed. For example, an applet cannot be used to extract all the links of the current user page. Another disadvantage of Java Applet is represented by the requirements of the Java Runtime Environment: first of all it is not available by default on all web browsers, moreover it has a large memory occupation (around 20 Mb) and applets cannot start until the Java Virtual Machine is running.

[4]http://www.sun.com/applets/

Flash[5] is a multimedia technology commonly used to create animations, to build interactive web pages and to develop client-side web applications. The flash files (called *Flash Movies*) run in a virtual machine called *Flash Player*, that is available for a wide variety of different browsers, platforms and devices. The Flash Player is smaller than Java runtime (less than 1 MB) and it is installed on over 500 million devices and more than 97% of Internet-enabled desktops[6]. Moreover, a Flash Player is embedded in many consumer electronics devices, like Kodak EasyShare-One digital camera: the user interface, built using Flash, enables simple navigation during picture taking and sharing, and includes rich graphical scene modes. Flash Movies can be programmed with a scripting language called *ActionScript*, that is an ECMAScript[7]–based programming language, object oriented, loosely–typed and has a syntax quite similar to C. In contrast with JavaScript (which is also ECMAScript compliant), ActionScript is compiled into bytecode which is interpreted by a virtual machine. ActionScript has a rich API supporting the elaboration of numbers, strings, XML and graphical element (vectorial and raster); it allows to play sounds and movies and to interact with server side application with a fast proprietary protocol (Flash Remoting[8]) or the slower SOAP (Simple Object Access Protocol).

AJAX (shorthand for Asynchronous JavaScript and XML) is not a technology in itself, but a term that refers to the use of a group of technologies together [16]. In fact, AJAX is a combination of JavaScript, DHTML (Dynamic HTML)[9], XML and the Remote Scripting (also described in [16]). Remote Scripting is used to deliver content dynamically without the need to refresh the page and DHTML is a method for creating interactive web pages by using a combination of a markup language (HTML) and a client–side scripting language (JavaScript): one major use of JavaScript is to write functions that are embedded in or included from HTML pages and interact with the Document Object Model (DOM). Other typical examples of JavaScript usage are: validating web form input, opening popup window, playing sounds, changing images size and performing text conversion operation. The scripts can be embedded in HTML pages or contained in *.js* files linked to the web pages. The overall AJAX web application model, compared to traditional web applications, is shown in Figure 5. Since JavaScript is an interpreted language, errors are not detected until the faulty program line is executed. Another problem of AJAX (and JavaScript in general) are the

[5]http://www.adobe.com/products/flash/
[6]NPD Online survey, conducted in April 2006
[7]http://en.wikipedia.org/wiki/ECMAScript
[8]http://www.adobe.com/products/flashremoting/
[9]http://www.w3.org/DOM/faq.html#DHTML–DOM, http://www.w3schools.com/dhtml/

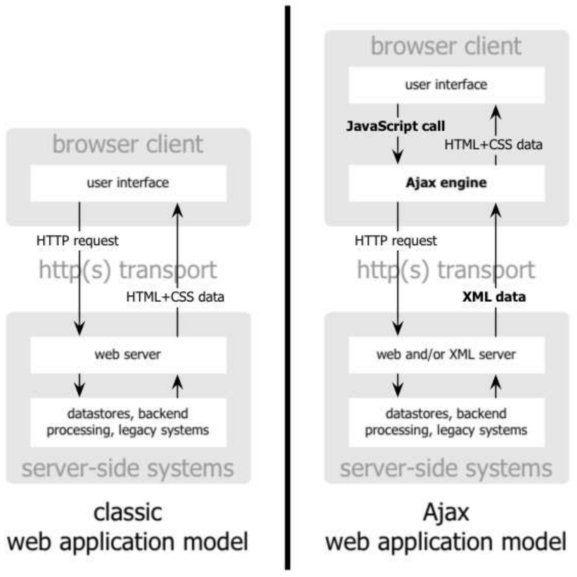


Fig. 5. The traditional web applications compared to the AJAX model. Figure by J. J. Garrett taken from [16].

differences between different JavaScript engine implementations, so applications must be tested systematically on the different target browsers and platforms. Nonetheless, AJAX is not only a scripting language that supports a rapid prototyping of web applications but it is also suitable for industry-strength systems (from WebGIS applications like Google Maps[10], to complex enterprise messaging and collaboration systems like Zimbra[11]).

To compare the different technologies, several sample applications that are available and freely accessible online can be evaluated. In particular several instant messengers have been implemented adopting Java Applet, Flash and AJAX technologies: for instance ICQ2Go![12] is is available both as a Java Applet and as a Flash application and Meebo[13] is developed with AJAX. Despite all are instant messenger applications, the user experience is very different: the Java version as ICQ2Go! has a very long startup time and it requires a huge amount of memory but it has most functions of the stand–alone ICQ client application and it is able to communicate with the server adopting the common ICQ protocol. The new Flash version of ICQ2Go! and Meebo are comparable in terms of user experience: both of them start much faster than the ICQ2Go! applet, but they still have a very good look and feel and an extensive set of functionalities. However, both the Flash and the AJAX version required a special server–side wrapper because they can communicate only with a XML protocol.

After the analysis of the various technologies, we have chosen to adopt AJAX in order to develop the Interface Agent. With AJAX, it is possible to create an agent hosted in the web browser that remains alive and active during the visualization of a web page. So it is possible to go beyond the classic web request/response model and develop proactive interface agents. We chose AJAX instead of Flash because it is possible to develop AJAX applications with Open Source tools (in fact, only a common text editor is needed). Today, a commercial IDE is required to build Flash web applications; although there is an Open Source ActionScript compiler[14], the lack of a proper full–featured Open Source IDE and mature tools for user interface drawing is a major drawback. Compared to Java Applet, instead, AJAX is lightweight and better integrated in the browsing environment: JavaScript functions have a complete control on the page content while applets are confined in a sandbox. This is a very important feature because the aim of an interface agent is to interact with the user, so an agent with more freedom of action over the interface can perform its task more effectively.

[10]http://maps.google.com/
[11]http://www.zimbra.com/
[12]http://go.icq.com/
[13]http://www.meebo.com/

B. The Interface Agent in the Overall Architecture

The interface agent starts its activity when a web page of the site is loaded into client Web Browser. The first action performed by the agent is adding to every link of the page a parameter (called *linkfrom*) with the URL of the current page as value. This action permits to identify the source page of every subsequent request. For example, assume that current page address is `http://host/index.html`, the link `<a href="events.html">Events</a>` included in the page will be rewritten as

```
<a href="events.html?linkfrom=index.html">Events</a>
```

Similarly, `<a href="news.jsp?news=1">Events</a>` will be rewritten as

```
<a href="news.jsp?news=1&linkfrom=index.html">Events
</a>
```

The content of the page is dynamically changed at client-side by JavaScript DOM (Document Object Model), so the original page on the server remains intact. DOM will allow scripts to dynamically access and update the content, structure and style of current page. The document can be further processed and the results of that processing can be incorporated back into the presented page. The agent doesn't update every link of the page, but only the HTTP links to the current site. So links to other sites, or links to a FTP repository or mail address remain unchanged.

The next action performed by the interface agent is to call the tracker. If the current page is called with the linkfrom parameter, this parameter is passed to the tracker. The tracker uses this parameter to build the traces. For example, if the URL of the current page is `events.html?linkfrom=index.html` the user's last page was `index.html`. The tracker can add a trace for the current user from index.html to events.html (or update an existing one). The tracker doesn't perform this operation itself, instead it informs the user agent on the MMASS environment, which is responsible for adding the trace. Then the interface agent can

[14]http://www.mtasc.org/

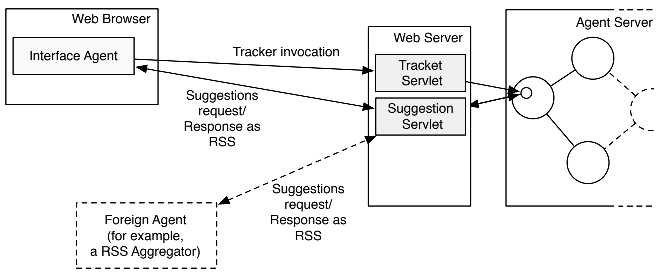

Web Browser
Interface Agent
Tracker invocation
Web Server
Tracket Servlet
Suggestion Servlet
Agent Server
Suggestions request/ Response as RSS
Suggestions request/ Response as RSS
Foreign Agent (for example, a RSS Aggregator)

Fig. 6. Interface Agent and Foreign Agent interaction with the MMASS user agent are performed through the Suggestion Servlet.

query the server to obtain the emerging links to be suggested to the user.Suggestions are in fact generated on the server–side and are published as an RSS[15](Really Simple Syndication) feed. The agent suggestion request is managed by the user agent (analogously as for traces). We choose RSS instead of a proprietary format because this allows foreign interface agents (other then our interface agent) to interact with the system.

The interface agents loads the RSS by using the `XMLHttpRequest`[16] class, which allows to perform an asynchronous request to the web server hosting the current web page and to store the response in a local variable. The response could be a XML document or plain text. In the first case, `XMLHttpRequest` stores the retrieved data in a DOM-structured object, which can be navigated using the standard JavaScript DOM access methods and properties, such as `getElementsByTagName()` and `childNodes[]`. The following code is an example of using `XMLHttpRequest` to asynchronously request the server side page `suggestions.jsp`:

```
req = new XMLHttpRequest();
req.onreadystatechange = processReqChange;
req.open("GET", "suggestions.jsp", true);
req.send(null);
```

In order to find out when the method has finished retrieving data, a specific event listener must be defined: in this case the method is `processReqChange`, reported in the following code snippet:

```
function processReqChange() {
 if ((req.readyState == 4) && (req.status == 200)) {
  // Gets the items from the XML document
  var xml = req.responseXML;
  var items = xml.getElementsByTagName("item");
  // Builds new suggestions
  var html = "";
  for (item in items) {
   var title = getValue(item, "title");
   var link = getValue(item, "link");
   // Adds a link and a carriage return
   html += "<a href='" + link + "'>" + title + "</a>";
   html += "<br/>";
  }
  // Replaces the content of the suggestions box
  document.getElementById("sBox").innerHTML = html;
 }
}
```

[15]http://www.rssboard.org/rss-specification
[16]http://www.w3.org/TR/XMLHttpRequest/

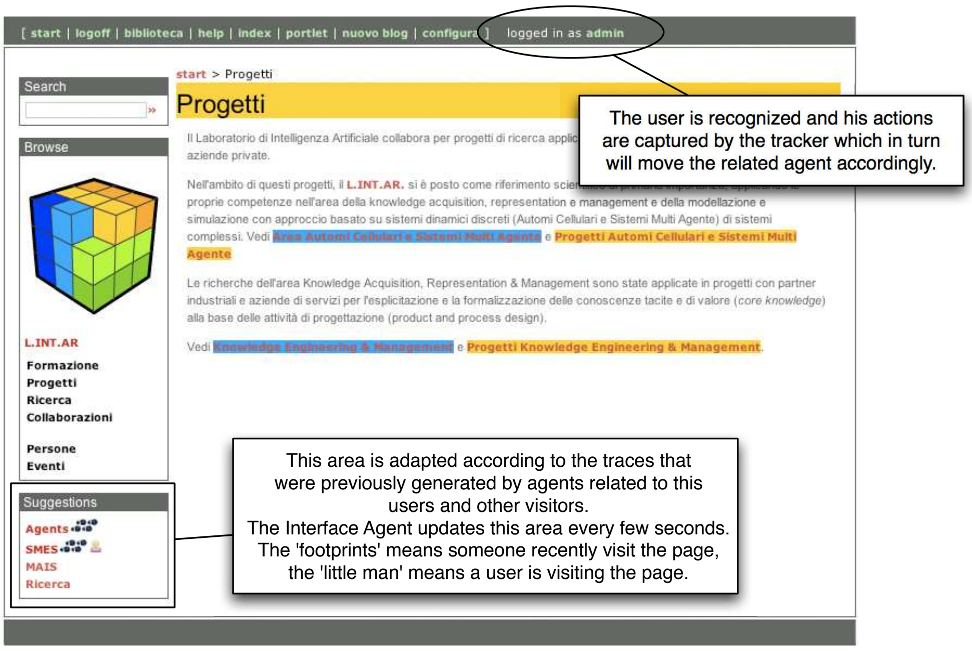
This method of the interface agent parses the RSS document and displays the suggestions in a box in the web page. This operation is done by using DHTML: the agent searches for the suggestion box (`sBox`) in the DOM of the page (which is a tree representation of the page HTML source) and than it replaces the content of the suggestion box with the freshly generated one. The latter is based on RSS suggestions: for each suggested page (represented as an item in the RSS) the Interface Agent adds a link to the page and uses the title of the page as label for the link. The following RSS is a suggestion example:

```
<?xml version="1.0" encoding="UTF-8"?>
<rss version="2.0"
  xmlns:lintar="http://www.lintar.disco.unimib.it/">
 <channel>
  <title>Suggested contents for index.html</title>
  <link>http://example.com/index.html</link>
  <language>en</language>
  <pubDate>Wed, 28 Jun 2006 02:28:19 +0200</pubDate>
  <ttl>1</ttl>

  <item>
   <title>Events</title>
   <link>http://example.com/events.html</link>
   <guid>http://example.com/events.html</guid>
   <lintar:usersTraces>75</lintar:usersTraces>
   <lintar:onlineUsers>3</lintar:onlineUsers>
  </item>

  [ ... more items ... ]

 </channel>
</rss>
```

In this example, the first suggested element is the Events page, whose URL is `http://example.com/events.html`. The tags in the `lintar` namespace are our extension to the basic RSS: the `<lintar:onlineUsers>` tag identifies the number of users currently viewing the page and the `<lintar:usersTraces>` tag represent the intensity of footprints on the page, in the spirit of [6]. Footprints are signs that one or more users have recently viewed the page. This information is also displayed by the interface agent on the suggestion box: the number of online users is displayed as a picture of little red man and the presence of users traces is represented by corresponding icon. The number of online users and the intensity of footprints are displayed in a tip box that it is shown when the mouse arrow is over the picture. It must be noted that the interface agent does not just provide a "one shot" behaviour. In fact, when initialized, it sets a timeout for a cyclical invocation of its main execution cycle by the web browser. In this specific application, in particular, it is this able to update and refresh the indication on the presence of other visitors and footprints on suggested pages. The overall cycle of interaction between the interface agent and the back end of the system is illustrated in Figure 6 and a screenshot of the web page enriched by the interface agent is shown in Figure 7.

C. The Adaptation Strategy

Every MAS agent of the implemented system provide personalized suggestions about items that user will find interesting, according to the history of the user and to the other


[start | logoff | biblioteca | help | index | portlet | nuovo blog | configura] logged in as admin

Search

Browse

L.INT.AR

Formazione
Progetti
Ricerca
Collaborazioni

Persone
Eventi

Suggestions

Agents
SMES
MAIS
Ricerca

start > Progetti

Progetti

Il Laboratorio di Intelligenza Artificiale collabora per progetti di ricerca applic
aziende private.

Nell'ambito di questi progetti, il L.INT.AR. si è posto come riferimento scie
proprie competenze nell'area della knowledge acquisition, representation e management e della modellazione e simulazione con approccio basato su sistemi dinamici discreti (Automi Cellulari e Sistemi Multi Agente) di sistemi complessi. Vedi Area Automi Cellulari e Sistemi Multi Agente e Progetti Automi Cellulari e Sistemi Multi Agente

Le richerche dell'area Knowledge Acquisition, Representation & Management sono state applicate in progetti con partner industriali e aziende di servizi per l'esplicitazione e la formalizzazione delle conoscenze tacite e di valore (*core knowledge*) alla base delle attività di progettazione (product and process design).

Vedi Knowledge Engineering & Management e Progetti Knowledge Engineering & Management.

The user is recognized and his actions are captured by the tracker which in turn will move the related agent accordingly.

This area is adapted according to the traces that were previously generated by agents related to this users and other visitors.
The Interface Agent updates this area every few seconds.
The 'footprints' means someone recently visit the page, the 'little man' means a user is visiting the page.

Fig. 7. A screenshot of a web page adapted according to gathered traces.

users path. These suggesting links have relationship with the previously introduced traces, which represent behaviors and movements of a user in a web site: the strategy which is adopted to select the most relevant traces to be presented to a given user considers the occurrence of trace generation and the success rate of the traces that were proposed.

A first element of this strategy is adopted when new users (or non authenticated ones) enter the site. In this case the user has no previous history (or it is not possible to correlate the user with his/her history), and the adopted strategy considers all stored traces, not considering the user which generated them. An additional information that is stored with traces is the number of times that the related trace was effectively selected and shown to a user and the number of times that the related link was effectively exploited by a user. This kind of information allows to obtain an indication of the success rate of the suggestions that were chosen by the agent, and can be exploited to select the traces to be shown in the adaptive block. When the agent has an indication of the user which issued the request, it may focus the selection activity to those traces that compose the history of user's activities in the web site, in a web customization framework. In fact traces include an indication of the agent which generated them, and in turn agents are related to registered users. Moreover, in order to focus on a specific user's history but do not waste the chance to exploit other users' experiences, just two of the three available slots for emergent links are devoted to traces that were generated by that user and one is selected according to the strategy adopted for anonymous or new users. Because the time spent on a page had a strong correlation with explicit interest [17], the adopted strategy uses this information to refine the proposed suggestions.

An example of page adaptation refers to the adoption of a recurrent trace leading from the index of the web site to a content page, that is not directly connected to the index but that is visited very frequently. This kind of "vertical"[17] emerging link is frequently observed in the prototypal implemetation of the system, which is installed in a web site presenting information about a research laboratory as well as information on courses held by members of the group[18]. Since the number of students of some of these courses is very high, they frequently generate traces connecting the index to the page related to those courses. These traces represent effective shortcuts allowing to bypass intermediate index pages related to education activities and university courses. However, emerging links can also connect pages deep in the site structure. For example, a page related to a project might not be explicitly connected to another page describing a particular modeling approach adopted in that project, but a user might browse the web site and effectively discover that page, causing the generation by the system of a correspondant trace connecting the project and the modeling approach. This trace might not be extremely relevant to all visitors of the web site, due to the fact that this navigation path will probably be not very frequent, but if the visitor is a registered user the trace could be stored and suggested anyway, since a number of slots in the adaptive area of the page is reserved to user–generated emerging links.

This strategy for the exploitation of the gathered and stored traces, based on users' behaviours and movement in the web site environment, represents a very simple way of exploiting this kind of information without requiring an off-line analysis

[17]Here vertical is intended as describing the typical navigation path starting from an index page and going deeper into the web site.

[18]http://www.lintar.disco.unimib.it

of the logs generated by the web server. The design, implementation and test of more complex strategies, for instance based on details of the outcomes of emerging link proposals (e.g. which user effectively followed the suggested adaptive hyperlink) are object of future works.

V. Related Work

There are several different approaches and relevant experiences in the area of web site adaptation, and some of them are also related to agent technologies. In particular, a relevant approach provides the adoption of information agents supporting users in their navigation [18]. These agents generally consider both the specific behaviour of the user and the actions of other visitors, and adopt multiple strategies for making recommendations (e.g. similarity, proximity, access frequency to specific documents).

The Footprints system [6] instead provides a site optimization through the metaphor of site visitors leaving traces in their navigation. These signals accumulate in the environment, generating awareness information on the most frequently visited areas of the web site. No user profile is needed, as visitors are essentially provided this information which could represent an indicator of the most interesting pages to visit. The metaphor of the structure of the web site as an environment on which visitors move in their search for information is very similar to the one on which the proposed framework is based, but we also propose the exploitation of the gathered information on users' paths for user specific customization. Another interesting recent work [19] represents an attempt to integrate interaction mechanisms similar to the one adopted by Footprints, often referred to as stigmergic interaction mechanisms [20], and cognitive agents. This line of research could represent an interesting way to integrate the proposed approach, which is able to generate and manage awareness contextual information, with higher level mechanisms and strategies of adaptation.

Other approaches provide instead the generation of index pages [3], that are pages containing links to other pages covering a specific topic. These pages, resulting from an analysis of access logs aimed at finding clusters grouping together pages related to a topic, are proposed to web masters in a computer-assisted site optimization scheme. A different approach provides the real-time generation of shortcut links [21], through a predictive model of web usage based on statistical techniques and the concept of expected saving of a shortcut, which considers both the probability that the generated link will be effectively used and the amount of effort saved (i.e. intermediate links to follow). In particular, this framework is very similar to the one proposed here with reference to the aims of the overall system, but it incorporates a complex algorithm for off-line analysis of logs, while the proposed approach provides a light and dynamic generation of most probable useful links and the storage of these proposals and high level information on site usage for a possible further off-line analysis.

A different approach to web site adaptation provides the adoption of a learning network to model the evolution of a distributed hypertext network, such as a web site [22]. Also in this case the adaptation provides a modification in the structure of a web site, and the concept of emergent link and the underlying mechanisms present a similarity with the learning rules adopted for that kind of learning network. However that approach also provides a deep modification in the architecture of the site and modifications in the web protocols, while this work aims at providing a solution that can be easily integrated with a traditional web architecture. Moreover, recent developments of that line of research were aimed at identifying analogies and relations among words by means of web mining [23], rather than realizing adaptive web systems.

The introduced system supporting web site adaptation seems more similar to a recommendation system. A relevant type of recommender exploiting users' behaviours to decide which contents could be interesting for a certain visitor is represented by the collaborative filter approach [24]. The latter has been adopted in different recommendation systems, filtering mail messages, newsgroup articles and web contents in general, but typically requires users to rate these items. Moreover, it generally provides a concept of explicit users descriptions through profiles which can be compared to determine similarity among them. The idea is that contents that received a high rating by a certain user could be considered interesting by a similar user. The introduced system instead does not require an explicit rating of contents, but it rather observes the frequency of specific navigation paths, and exploits emergent links for customization or optimization of site structure. However, the adaptive block of the page can include emerging links that are not related to the specific visitor who is currently browsing that page, but were generated by other users which frequently followed paths that the current one still did not follow. From this point of view, the system provides a very basic collaborative browsing scheme, but a more through analysis of a possible integration with this approach is object of current and future works.

VI. Conclusions and Future Developments

This paper introduced a general framework providing the adoption of a web site as an environment on which agents related to visitors move and possibly interact. This approach allows the gathering of a structured form of information on users' behaviours and activities in the web site. The concept of emerging links and traces have been introduced in order to support an application exploiting information on users' browsing history for sake of web pages adaptation. The introduced framework and the application to web site adaptation have been designed and implemented, exploiting a platform supporting systems based on the MMASS model.

A campaign of tests aimed at evaluating the effectiveness of the adaptation approach, and also for sake of tuning the involved parameters (e.g. timings, number of presented possible emerging links) is under way. This evaluation will be based on user interviews and also on the exploitation of the gathered information of the success rate of proposed adaptive hyperlinks. Such an indicator might be obtained as a ratio between the number of times an emerging link has been actually selected by a user and the total number of times its has been shown. However, it must be noted that we currently do not have an indication of threshold to discriminate successful suggestions from unsatisfactory ones; a further analysis of methods adopted to evaluate related approaches is currently being carried out. The results of this evaluation might also lead to consider the modelling, design and implementation of more complex trace selection strategies, and thus a more complex behaviour for the interface agent.

Future works will be focused on the introduction and exploitation of higher level semantic information related to the site structure and contents, and thus agents' environment, aimed at providing additional forms of adaptation, including images and multimedia contents. While in [25] an analysis on how a conceptual view on the topics may be used as an additional level of description of the environment, another aspect that will be considered is the possibility to improve the effectiveness of web–based applications supporting processes with adaptive functionalities. Finally, a further development provides also the design and implementation of a prototype supporting the context-aware interaction among web site visitors. In this framework, the environment related to the web site also supports the mutual perception of the agents situated in it and it also supports a form of interaction among them depending on their relative positions. The latter can be thus considered as a form of context–dependant interaction. A more thorough analysis of the possible applications of this approach can be found in [25], and a prototypal implemetation of these interaction mechanisms is currently under way.

References

[1] A. S. Tanenbaum, *Computer Networks - third edition*. Prentice Hall, 1996.

[2] M. Perkowitz and O. Etzioni, "Adaptive Web Sites: an AI Challenge." in *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI 1997)*, 1997, pp. 16–23.

[3] ——, "Adaptive Web Sites," *Communications of the ACM*, vol. 43, no. 8, pp. 152–158, 2000.

[4] R. Cooley, "The Use of Web Structure and Content to Identify Subjectively Interesting Web Usage Patterns," *ACM Transactions on Internet Technology*, vol. 3, no. 2, pp. 93–116, 2003.

[5] D. Weyns, H. V. D. Parunak, F. Michel, T. Holvoet, and J. Ferber, "Environments for Multiagent Systems State-of-the-art and Research Challenges." in *Environments for Multi-Agent Systems, First International Workshop (E4MAS 2004)*, ser. Lecture Notes in Computer Science, vol. 3374. Springer–Verlag, 2005, pp. 1–47.

[6] A. Wexelblat and P. Maes, "Footprints: History-Rich Tools for Information Foraging," in *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM Press, 1999, pp. 270–277.

[7] J. Liu, S. Zhang, and J. Yang, "Characterizing Web Usage Regularities with Information Foraging Agents," *IEEE Transactions Knowledge and Data Engineering*, vol. 16, no. 5, pp. 566–584, 2004.

[8] D. Helbing, F. Schweitzer, J. Keltsch, and P. Molnár, "Active Walker Model for the Formation of Human and Animal Trail Systems," *Physical Review E*, vol. 56, no. 3, pp. 2527–2539, January 1997.

[9] R. C. Miller and K. Bharat, "Sphinx: a Framework for Creating Personal, Site-specific Web Crawlers," *Computer Networks and ISDN Systems*, vol. 30, no. 1–7, pp. 119–130, 1998.

[10] D. Weyns, F. Michel, and H. V. D. Parunak, Eds., *Environments for Multi-Agent Systems, First International Workshop (E4MAS 2004)*, ser. Lecture Notes in Artificial Intelligence, vol. 3374. Springer–Verlag, 2005.

[11] D. Weyns and T. Holvoet, "Model for Simultaneous Actions in Situated Multi-Agent Systems," in *First International German Conference on Multi-Agent System Technologies, MATES*, ser. Lecture Notes in Computer Science, vol. 2831. Springer–Verlag, 2003, pp. 105–119.

[12] M. Mamei, F. Zambonelli, and L. Leonardi, "Co-fields: Towards a Unifying Approach to the Engineering of Swarm Intelligent Systems," in *Engineering Societies in the Agents World III: Third International Workshop (ESAW2002)*, ser. Lecture Notes in Artificial Intelligence, vol. 2577. Springer–Verlag, 2002, pp. 68–81.

[13] K. Hadeli, P. Valckenaers, C. Zamfirescu, H. V. Brussel, B. S. Germain, T. Hoelvoet, and E. Steegmans, "Self-organising in Multi-Agent Coordination and Control Using Stigmergy," in *Engineering Self-Organising Systems: Nature-Inspired Approaches to Software Engineering*, ser. Lecture Notes in Computer Science, vol. 2977. Springer–Verlag, 2004, pp. 105–123.

[14] S. Bandini, S. Manzoni, and C. Simone, "Dealing with Space in Multi–Agent Systems: a Model for Situated MAS," in *Proceedings of the first international joint conference on Autonomous agents and multiagent systems*. ACM Press, 2002, pp. 1183–1190.

[15] S. Bandini, S. Manzoni, and G. Vizzari, "Towards a Platform for Multilayered Multi Agent Situated System Based Simulations: Focusing on Field Diffusion," *Applied Artificial Intelligence*, vol. 20, no. 4–5, pp. 327–351, 2006..

[16] J. J. Garrett, "AJAX: a New Approach to Web Applications," Adaptive Path Essay, Tech. Rep., 2005. [Online]. Available: http://www.adaptivepath.com/publications/essays/archives/000385.php

[17] M. Claypool, P. Le, M. Waseda, and D. Brown, "Implicit Interest Indicators." in *Intelligent User Interfaces*, 2001, pp. 33–40.

[18] M. J. Pazzani and D. Billsus, "Adaptive Web Site Agents," *Autonomous Agents and Multi-Agent Systems*, vol. 5, no. 2, pp. 205–218, 2002.

[19] A. Ricci, Omicini, M. Viroli, L. Gardelli, and E. Oliva, "Cognitive Stigmergy: a Framework Based on Agents and Artifacts," in *3rd International Workshop "Environments for Multi-Agent Systems" (E4MAS 2006)*, D. Weyns, H. V. D. Parunak, and F. Michel, Eds., 2006, pp. 44–60.

[20] G. Theraulaz and E. Bonabeau, "A Brief History of Stimergy," *Artificial Life*, vol. 5, no. 2, pp. 97–116, 1999.

[21] C. R. Anderson, P. Domingos, and D. S. Weld, "Adaptive Web Navigation for Wireless Devices," in *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI 2001)*, 2001, pp. 879–884.

[22] J. Bollen and F. Heylighen, "Algorithms for the Self-Organisation of Distributed, Multi-User Networks. Possible Application to the Future World Wide Web," in *Proceedings of the 13th European Meeting on Cybernetics and Systems Research*, R. Trappl, Ed. Austrian Society for Cybernetic Studies, 1996, pp. 911–916.

[23] F. Heylighen, "Mining Associative Meanings from the Web: from Word Disambiguation to the Global Brain," in *Proceedings of the International Colloquium: Trends in Special Language & Language Technology*, R. Temmerman and M. Lutjeharms, Eds. Standaard Editions, Antwerpen, 2001, pp. 15–44.

[24] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl, "Grouplens: an Open Architecture for Collaborative Filtering of Netnews," in *CSCW '94: Proceedings of the 1994 ACM conference on Computer supported cooperative work*. ACM Press, 1994, pp. 175–186.

[25] S. Bandini, M. Sarini, C. Simone, and G. Vizzari, "WWW in the Small: Towards Sustainable Adaptivity," *World Wide Web Journal*, 2006 (to appear).

MAST: an Agent Framework to Support B2C E-Commerce

Salvatore Garruzzo, Domenico Rosaci and Giuseppe M.L. Sarné
DIMET - University Mediterranea of Reggio Calabria
Località Feo di Vito - 89060 Reggio Calabria, Italy
Email: {salvatore.garruzzo, domenico.rosaci, sarne}@unirc.it

***Abstract*— In this paper we present an XML-based multi-agent system, called *Multi Agent System for Traders* (MAST), that completely supports Business-to-Customer E-commerce activities, including advertisements and payments. MAST helps both customers and merchants in their tasks with a homogeneous and personalized approach. In particular, E-payments in MAST are implemented under the availability of financial institutions. This avoids exchanging of sensible customers' information and reinforces the confidence between customers and merchants. A complete prototype of MAST has been implemented in the JADE framework, and it has been exploited for realizing some experiments, in order to evaluate its performances.**

I. Introduction

A great contribution to the Internet diffusion has been provided by E-Commerce (EC) activities, i.e. all trading activities carried out by means of Internet. In [15] a classification of the EC activities in homogeneous categories has been realized on the basis of the typology of the traders and of the specific trade activity carried out over the Internet. In this paper we deal with one of these categories, i.e. the Business-to-Consumer (B2C), that can be compared to the retail trade of traditional commerce.

Nowadays, the B2C involves a large number of merchants interested in offering products using a convenient media and customers that desire to purchase those products. In this context, customers and merchants can exploit different opportunities as: *(i)* absence of time and space boundaries; *(ii)* simple, fast and comfortable purchases; *(iii)* low costs and several sale terms available. However, a significant customer-merchant distrust still persists, mostly due to the absence of personal contacts and to a low acceptance of the e-payment methods for security reasons. To capture the different phases carried out by enacting a B2C process, some behavioural models can be exploited, and particularly, in this paper we adopt the Consumer Buying Behavior (CBB) model [8], where the trading activities have been embedded in six different phases (resp., "Need Identification", "Product Brokering", "Merchant Brokering", "Negotiation", "Purchase and Delivery", "Service and Evaluation").

This work presents a multi-agent framework to support B2C activities of merchants and customers. Such a framework, called *Multi-Agent System for Traders* (MAST), is composed of a set of agents and a central agency. In particular, in MAST each merchant and each customer is provided by a software agent, managing a personal profile, able to support B2C activities during all the CBB stages. The MAST framework presents the following important features: *(i)* software agents are XML-based to manage agent profiles and messages in a light and easy manner and to realize agent communications in ACML language [3], [7] assuring portability; *(ii)* an *Ontology* is used as a common language for all agents allows to unify the representation of products and categories belonging to various catalogues; *(iii)* an e-payment protocol called AIPP (Agent Internet Payment Protocol) [6], based on existing financial institutions, is fully compliant with the standard FAST [2] and it is used together with single-use account identifiers [18]; *(iv)* a central agency provides agents with some services and cooperates with them to realize only the "Need Identification" and the "Service and Evaluation" stages of the CBB model in an efficient way.

The paper is organized as follows: the MAST framework is presented in the following section. In Sect. III the AIPP protocol is briefly illustrated and in Sect. IV the adopted functionalities for customer and merchant support are described. In Sect. V the MAST prototype and performances are discussed. Section VI deals with some Related Work and finally, in Sect. VII, some conclusions are drawn.

II. The MAST Framework

In the MAST framework, represented in Fig. 1, each customer C and merchant M is associated to her/his personal agent (resp., c and m) and with her/his financial institution (FI). All agents are logged into the MAST Agency (Ag). Both agents and agency support B2C activities managing (in terms of insertion, deletion and updating) their respective *Knowledge* profiles. In this section, agents and agency will be briefly described by illustrating their profiles and behaviours, while the B2C support activities in the CBB stage are exposed in Sect. IV.

A. The MAST Agents

In the following, U denotes the generic user (a customer or a merchant) and a represents her/his agent. Each MAST agent manages its Agent Knowledge (AK) profile, represented in Fig. 2 and described by the following elements:

- UD (*User Data*), contains the user's name ($Name$) and address ($Address$), login identifier (AcL), password (AcP), real (Ac) and single-use (AcT) user's account

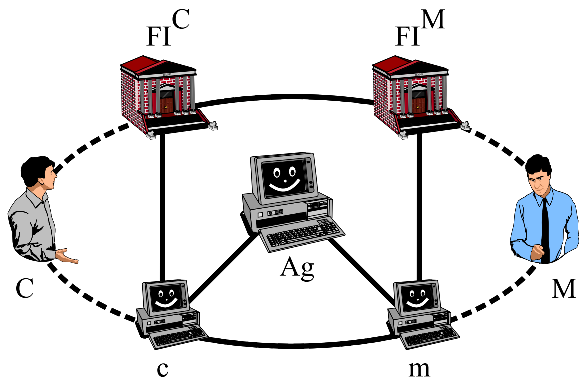


Fig. 1. The MAST architecture

identifiers, all referred to the user's account into FI. Note that Ac and AcT include also FI coordinates.

- AD (*Agent Data*), containing the user's agent (aI) and Agency (AgI) identifiers and a pruning threshold (T) to delete uninteresting information from AK.
- O (*Ontology*). In order to identify products of interest for the users, in our framework as Ontology we adopt the North America Industry Classification (NAICS) [13], which is an official hierarchical industrial classification used in North America, employing unified evaluation criteria. In MAST, the 6 digits NAICS code is used to identify a product. It is clear that other ontologies of this kind might as well be adopted instead of the NAICS coding.
- PD (*Product Data*) is a set of products, where each product is represented by an identifier (PI) and described by the following elements: NAICS code (N); model (M); brand (B); price (P); currency (C); commercial unit (U); auction flag (A) that is set to 1 if the product has a fixed price otherwise it is set to 0; tax (X); benefits set ($BSet$), eventually empty, of added values; delivery set ($DSet$), that collects the delivery identifiers (see DD section) related to the product.
- DD (*Delivery Data*) is a set of elements, where each element, represented by an identifier (DI), is described by the delivery time (DT) and by the fixed (F) and variable (V) costs. Note that DD collects the data of the chosen delivery for a customer, while it collects the data of the delivery he/she makes available for a merchant.
- ADB (*Agent Data Base*) is a set of agent (and Agency) data where each element, represented by its identifier aI, is described by its Internet address (aA) and by the date of its update (aAU).
- UP (*User Profile*) is a set of data that an agent a obtains monitoring the CBB activities in the MAST environment; for a customer its c agent collects the data of the products which the customer is interested in; elsewhere, for a merchant its m agent collects the data of the CBB activities carried out in the site by the agents of the various customers for the products offered by the merchant. Each element of UP is represented by the identifier PI (the same of the PD section) and it is described by the following elements: visit counter (VC); first visit (FV), one before the last visit (PV) and last visit (LV) dates; product rate (R); a set ($PASet$), where each element is associated to an agent that has been interested in the product, and that is composed by an agent identifier (aI), the highest CBB stage reached and eventually the delivery identifier (DI) and the auction flag (A). More in detail, the rate R represents the interest of a customer for a specific product and it is updated (by a) when a CBB activity is monitored by a using the following formula:
$$R = \phi \cdot \sum_{\xi=1}^{5} \left[\frac{CV \cdot (1+PV-FV)}{((6-\xi)^2 \cdot (1+LV-PV))} \right]$$
where: $\xi \in [1, \cdots, 5]$ identifies one among the first five CBB stages; $\phi \in \{1; -1\}$ describes the satisfaction of the customer about a product (ϕ is usually set to 1, but it is set to -1 by the customer only if in the last and optional CBB stage he/she is unsatisfied of the purchased product; for a merchant, ϕ is always set to 1). Furthermore, the differences among FV, PV and LV are expressed in days[1] beginning from FV. Note that R depends on the number of times that an activity has occurred in a CBB stage, the relevance of the involved CBB stage and the more recent accesses.

The information in the described structures are used by an agent a to realize its goals, as explained in the following, excluding the CBB support which is presented in Sect. IV. More in detail:

- ***setup steps***: semi-automatic procedures are activated to: *(i)* set initially or update UD, AD, ADB, interacting with Ag when it is needed as in the first a's activation; *(ii)* remove a from the system for an U's request to Ag.
- ***operational steps***: a customer agent is automatically activated (resp., deactivated) when a Web session starts (resp., ends "per se" or for an explicit customer's choice), whereas a merchant agent is automatically activated (resp., deactivated) when its site is on-line (resp., off-line or for an explicit merchant's choice). An agent performs the following activities: *(i)* it sends periodically to Ag its aA; *(ii)* it constructs its profile to support its user updating its AK w.r.t. each agent contact and each access for a product in one or more CBB stages; *(iii)* in order to realize the first phase of the "Need Identification" CBB stage, in MAST a customer agent periodically sends to Ag a list (L) containing the NAICS code of those products that meet interests and preferences of its user, ordered on the basis of their rate R; *(iv)* periodically each agent prunes its AK from some evaluated unimportant information on the basis of the values of the rating w.r.t. the threshold T.

B. *The MAST Agency*

The Agency Knowledge (AgK) profile is described by:

- AgD (*Agency Data*), that is composed by the Agency Identifier (AgI) and Internet Address (AgA).

[1]The choice of the day as reference time unit is due to the characteristics of the problem, given that purchases usually do not occur often in time (e.g., each minute or hour). However, it is possible to change the reference time unit without influencing the generality of the model.

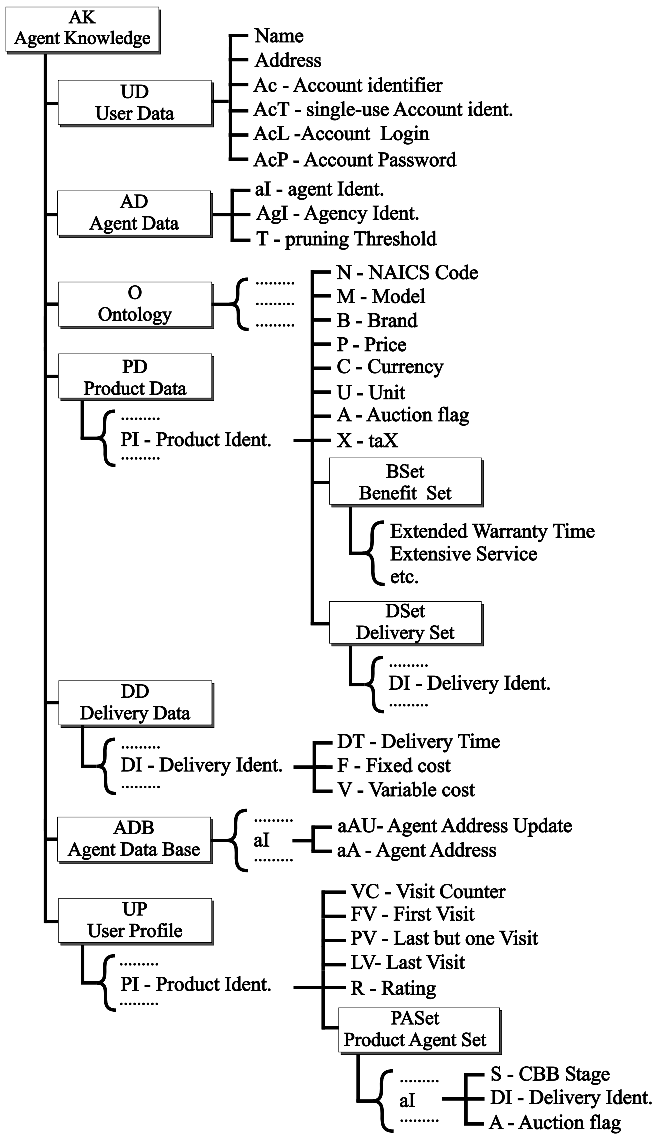


Fig. 2. The Agent Knowledge (AK)

- ADB (*Agent Data Base*), that is a set of agent data where each element, represented by its identifier aI, is described by: an Internet address (aA); the date of the aA update (aAU); a list (L) of NAICS code referred to those products of interest or preferences; the name ($Name$) and address ($Address$) of the agent's owner.
- aPT (*Agent Pruning Threshold*) that is exploited to deallocate long-time inactive agents.

The behaviour of the Agency consists of:

- ***affiliate managing steps***: The Agency carries out the following operations automatically: *(i)* when it is required, the Agency affiliates an agent sending it its *Identifier* and at this point the agent is logged and operative; *(ii)* the Agency updates the agent data when it changes; *(iii)* the Agency stores, for each active agent, the current address and the list L that the agent periodically sends to Ag; *(iv)* if a user requires an agent deletion to the Agency or if an agent is inactive for a time longer than the pruning threshold aPT, then the Agency deletes the agent and informs the community.
- ***service managing steps***: The Agency provides some services to agents, namely: *(i)* the support to realize some CBB stages efficiently as will be described in the following; *(ii)* a broadcasting message service (e.g., to provide an agent m offer to all agents c); *(iii)* a yellow page service, where each affiliate can ask the address of another MAST affiliate.

III. The Agent Internet Payment Protocol (AIPP)

Payment schemes can be assessable with subjective criteria, as customer acceptance or trust [14], and objective criteria, as functionality and quality parameters like transaction cost, security, privacy, etc. The presence of a network in a payment scheme introduces new issues, absent in traditional scenarios [1], [16], where: *(i)* identities of the transaction actors need to be authenticated and validated; *(ii)* payments and their effects guaranteed; *(iii)* operations, frauds and legal risks minimized. In addition, an extended use of standard protocols, existing products and services, payer anonymity, purchases confidentiality and low costs are desirable.

Currently, the most used e-payment system is the credit card, but in this case a credit card number should be provided to the merchant; this could be risky because the card number is provided over Internet and/or stored in the merchant site. The electronic cash systems cannot be used due to law and crime prevention regulation/legislation [1]. Recently, centralized account schemes have grew quickly in popularity for their aptitude to integrate usual financial instruments in a secure Internet transaction context. This payment family, also proposed by well known financial institutions, includes general purpose or e-commerce specific applications and can be realized completely either in secure software or in secure hardware.

A centralized account approach has been proposed in 1999 by the Financial Service Technology Consortium (FSTC) with the Financial Agents Secure Transaction (FAST) project [5]. The FAST team has developed five payment schemes for different scenarios (without specifying any detailed protocol) based on financial institutions that manage user's accounts and agent technologies to take advantage from existing infrastructures. The main benefits are: *(i)* customers and merchants with no common authentication mechanisms (FAST is not an authentication model) are reciprocally authenticated by their financial institutions when they log in their on-line accounts with the usual procedures (commonly with login and password over an SSL connection [4]); *(ii)* payments occur directly via financial institutions to guarantee effective funds availability, funds transfer and connected effects, but also promote credit-push; *(iii)* interoperability among accounts located in different financial institutions is easy to effect (as between two banks) choosing among different transfer modalities usually available. On the contrary, it is hard when accounts are located into competitor payment systems; *(iv)* payments are carried out by agents that replace customers and merchants in most uninteresting and/or complex tasks.

The risks in FAST can be further minimized by transferring funds over interbanking networks, assigning to each message a time to live and a unique identifier, managing as much sensible information as possible off-line, etc. The problems of security communication among financial institutions, as those related to defense against viruses or hacker attacks, are beyond the FAST project objectives.

In this paper, we exploit the e-payment protocol AIPP (Agent Internet Payment Protocol) [6], complying with the FAST "pre-negotiation" scheme [2], together with single-use account identifiers [18]. In AIPP a low amount of information is exchanged without any explicit encryption level. Moreover, AIPP adopts only asynchronous agent communications without multiple Internet connections (other parties connected to the infrastructure, such as Internet providers, are considered as external risk factors). In this way, it is proposed as a potentially well acceptable Internet financial transaction method able to satisfy all issues of an e-payment scheme that have been previously described.

IV. The MAST Support to CBB Activities

MAST provides a support, in accordance to the CBB model, to customers and merchants in their EC activities. In MAST, typical interaction between agents involves a customer (C) with her/his agent (c) and financial institution (FI^C), a merchant (M) with her/his agent (m) and financial institution (FI^M), the Agency (Ag), a product (G) offered by a M. The appropriate financial institution typologies are limited to banks, card issuers or relevant financial organizations; further, it is assumed that payers and payees can manage their on-line accounts. In the following, the terms product and service will be used interchangeable.

In MAST, to avoid possible attacks, single-use account identifiers (preserving also financial privacy) and a nonce (i.e., an agent sender marker) are adopted, and a Time To Live (TTL) is used as message deadline for each agent communication. Moreover, to promote trust among customers and merchants, the AIPP protocol allows the FIs to be third parties in a financial transaction, still guaranteeing user's privacy.

Notation and data contents of the messages used in MAST to transfer in a consistent and efficient way the business information are illustrated before describing the MAST protocol. Note that the subscripts identify sender and receiver while *data* is an XML document[2], whose content is context sensitive (see Table I). More in detail:

- $INF_{x,y}(data)$: it requires/provides commercial information about a product;
- $REQ\_INV_{c,m}(data)$: it requires an invoice for a product offered by M;
- $INV_{m,c}(data)$: it contains the invoice required with $REQ\_INV_{c,m}(data)$;
- $PO_{c,m}(data)$: it is the purchase order w.r.t. $INV_{m,c}(data)$;

[2]MAST agents employ the eXtensible Markup Language (XML) [22] to overcome several heterogeneity problems (platforms, languages, applications and communication modalities) and to transfer business information in a consistent way. Note that specific agreements must be established on the tag semantic.

TABLE I
MESSAGE SPECIFICATION

Message	Message Content
$INF_{x,y}$	$\mathbf{H}(aS^x, aR^x, nc^x, TTL^x)$, $\mathbf{G}(N, M, B, P, C, U, A, X, BSet, DD, CUR, FP)$
$REQ\_INV_{c,m}$	$\mathbf{H}(aS^c, aR^c, nc^c, TTL^c)$, $\mathbf{G}(PI^M, N, M, B, P, C, U, A, X, BSet, DI^M, CUR, FP)$
$INV_{m,c}$	$\mathbf{H}(aS^m, aR^m, nc^m, TTL^m, PII^m), \mathbf{G}(PI^M, N, M, B, P, C, U, A, X, BSet, DI^M, DD, F, V, CUR, FP), \mathbf{F}(FII^M, FIA^M, AcT^M)$
$PO_{c,m}$	$\mathbf{H}(aS^c, aR^c, nc^c, TTL^c, PII^m)$, $\mathbf{F}(FII^C, FIA^C, AcT^C, Address^C)$
$PE_{x,y}$	$\mathbf{H}(aS^x, aR^x, nc^x, TTL^x, PII^m)$
$PA_{x,y}$	$\mathbf{H}(aS^x, aR^x, nc^x, TTL^x, PII^m)$
MTO_{c,FI^C}	$\mathbf{H}(aS^c, aR^c, nc^c, TTL^c, PII^m)$, $\mathbf{F}(FII^M, FIA^M, AcT^M, H(INV_{m,c}))$
$A\_MTO_{FI^C,c}$	$\mathbf{H}(aS^{FI^C}, aR^{FI^C}, nc^{FI^C}, TTL^{FI^C}, PII^m)$
$R\_MTO_{FI^C,c}$	$\mathbf{H}(aS^{FI^C}, aR^{FI^C}, nc^{FI^C}, TTL^{FI^C}, PII^m)$
$ACT\_COD_{x,y}$	$\mathbf{H}(aS^x, aR^x, nc^x, TTL^x, PII^m), \mathbf{F}(H(INV_{m,c}))$
$NEW\_AI_{x,y}$	$\mathbf{H}(aS^x, aR^x, nc^x, TTL^x), \mathbf{F}(AcT^y)$
$EVAL_{c,m}$	$\mathbf{H}(aS^c, aR^c, nc^c, TTL^c, PII^m)$

In the first three CBB stage the messages can be addressed to c agents chosen among those listed in ADB or employing the Ag's broadcasting messages service

- $PE_{x,y}(data)$ (resp., $PA_{x,y}(data)$): it notifies that the payment has been performed (resp., aborted) w.r.t. $PO_{c,m}(data)$;
- $MTO_{c,FI^C}(data)$: it is an irrevocable money transfer order w.r.t. $INV_{m,c}(data)$;
- $A\_MTO_{c,FI^C}(data)$ (resp., $R\_MTO_{c,FI^C}(data)$): it notifies the MTO acceptance (resp., rejection) w.r.t. $MTO_{c,FI^C}(data)$;
- $ACT\_COD_{x,y}(data)$: it contains the MTO activation code w.r.t. $INV_{m,c}(data)$;
- $NEW\_AI_{x,y}(data)$: it contains a new single-use account identifier to be employed in the next purchase or sell;
- $EVAL_{c,m}(data)$: it is an optional evaluation of a purchase.

A *data* XML document is structured in three sections including:

1) *H (Header)* that is composed by: agent identifiers of *Sender* (aS) and *Receiver* (aR); *CBB Stage* (S); *Nonce* (nc) that is an agent's marker; *Time To Live* (TTL); *Product Invoice Identifier* (PII).
2) *G (Products)* that encodes: *Product Identifier* (PI) and the product data ($N, M, B, P, C, U, A, X, BSet$) previously described in Sect. II-A; one or more *Delivery Identifiers* (DI) with the corresponding data (DD, F, V), previously described in Sect. II-A; *Commercial Unit Required* (CUR); *Final Price* (FP).
3) *F (Financial)* is constituted by: *Financial Institution Identifier* (FII); *Financial Institution Internet Address* (FIA); *Financial Institution Single-Use Account identifier* (Ac); *User Address* ($Address$).

The actions performed by agents in MAST to support customers and merchants in their B2C activities during all CBB stages are described below in detail for each CBB stage and represented in Fig. 3.

a) Need Identification Support: ($\xi = 1$). In the first CBB stage, customers identify their needs and merchants advertise their offered products (G) to as more potential customers as possible. In detail: *(i)* when an M wants to make an offer about a product to potential Cs, he/she has to submit own offer sends to Ag an $INF_{m,c}$; *(ii)* in a first phase the Ag, on the basis of the lists L provided by the c agents, takes care of sending the offer to the potentially appropriate c agents, then in a second phase the c agents present such offers to their Cs only if they are fully compatible with their interests and preferences.

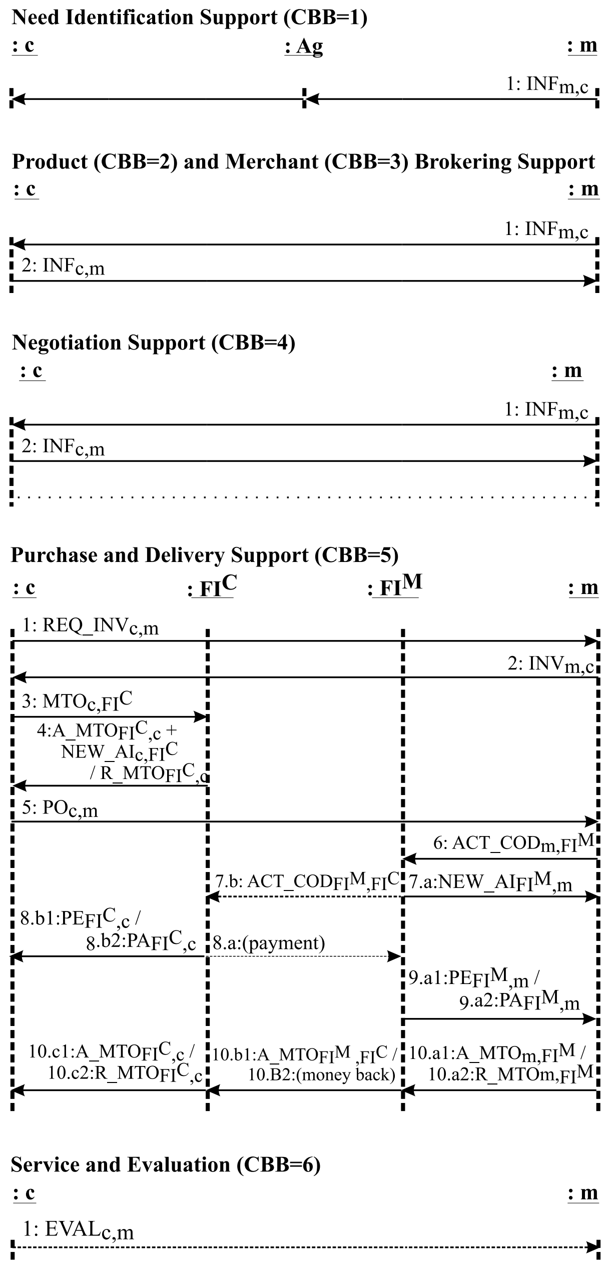
b) Product Brokering Support: ($\xi = 2$). This stage occurs when a customer has identified a need and looks for a suitable product to satisfy it. In detail: *(i)* C can ask information on the desired product typology to one or more Ms by means of $INF_{c,m}$; *(ii)* all Ms that have a product that matches the C request, reply with a new $INF_{m,c}$ with all the details of the products and commercial information.

c) Merchant Brokering Support: ($\xi = 3$). A customer identifies the most suitable merchant to purchase a product carrying out the following actions: *(i)* If C has sufficient knowledge of the product details, a c's message $INF_{c,m}$ is sent to one or more merchants; *(ii)* if there is a product that matches C's request, M replies with a message that reports a complete description of the product; in such a way c can select the best product offer. Note that if in the previous stage C has received a sufficient number of $INF_{m,c}$ it is possible to choose a merchant without carrying out this present stage explicitly.

d) Negotiation Support: ($\xi = 4$). In this stage a pair of customer and merchant define the purchase details. They realize suitable strategies in a multi-round session for their respective bids and offers presented by means of messages. This stage is closed when an agreement is reached or the timeout TTL of the last message has elapsed.

e) Purchase and Delivery Support: ($\xi = 5$). In this stage the customer purchases, pays and chooses a delivery modality for a product offered by a merchant employing the AIPP protocol where: payer and payee identities are authenticated by their respective financial institutions during their on-line accounts accesses (usually with login and password over a *SSL Internet session*); payments occur directly among the financial institutions; Single-use account identifiers are adopted; No heavy protocol is needed; no sensible financial and commercial information is exchanged to assure privacy; financial institutions are third parties in the transaction to guarantee customers and merchants. The actions performed in this stage are: *(i)* When C wants to purchase a product offered by M, he/she sends the message $REQ\_INV_{c,m}$; *(ii)* m replies with $INV_{m,c}$ (a pro-forma invoice); *(iii)* c logs into FI^C and then orders a *Money Transfer Order* (MTO_{c,FI^C}) to FI^M payee; *(iv)* FI^C accepts/rejects the MTO on the basis of the existence of sufficient C's funds and notifies to c its choice with a $A\_MTO_{FI^C,c}$ + a new single-use account

identifier (AcT) for the next purchase or with a $R\_MTO_{FI^C,c}$ message; *(v)* c sends a $PO_{c,m}$ to effect the purchase order; *(vi)* m logs into FI^M and sends the required payment activation code ($H(INV_{m,c})$ to FI^M; *(vii)* FI^M provides M with a new single-use account identifier (AcT) for the next sell and sends to FI^C the payment code ($H(INV_{m,c})$; *(viii)* if the activation code is the same as that provided by c, then FI^C effects the payment via FI^M and informs c about the state of success ($PE_{FI^C,c}$) or failure ($PA_{FII^C,c}$) of the MTO process; *(ix)* if the payment has been performed by FI^C,



Need Identification Support (CBB=1)
: c
: Ag
: m
1: INFm,c
Product (CBB=2) and Merchant (CBB=3) Brokering Support
: c
: m
1: INFm,c
2: INFc,m
Negotiation Support (CBB=4)
: c
: m
1: INFm,c
2: INFc,m
Purchase and Delivery Support (CBB=5)
: c
: FIC
: FIM
: m
1: REQ_INVc,m
2: INVm,c
3: MTOc,FIC
4:A_MTOFIC,c + NEW_AIc,FIC / R_MTOFIC,c
5: POc,m
6: ACT_CODm,FIM
7.b: ACT_CODFIM,FIC
7.a:NEW_AIFIM,m
8.b1:PEFIC,c / 8.b2:PAFIC,c
8.a:(payment)
9.a1:PEFIM,m / 9.a2:PAFIM,m
10.c1:A_MTOFIC,c / 10.c2:R_MTOFIC,c
10.b1:A_MTOFIM,FIC / 10.B2:(money back)
10.a1:A_MTOm,FIM / 10.a2:R_MTOm,FIM
Service and Evaluation (CBB=6)
: c
: m
1: EVALc,m

Fig. 3. UML of the MAST support activities

then FI^M informs m with a $PE_{FI^M,m}$ message, otherwise after the TTL of the $ACT\_COD_{FI^M,FI^C}$ message FI^M informs m with a $PA_{FI^M,m}$ of the sell failure; *(x)* finally, m could however accept the payment informing FI^M, and consequently FI^C, or refuse it aborting the sale and returns back the money to FI^C by means of its FI^M. At last FI^C will inform c whether the product has been purchased or not.

f) Service and Evaluation: ($\xi = 6$). It is an optional feedback provided by a customer to express her/his dissatisfaction about the purchase of a product, the merchant or both. Two kinds of actions can be carried out by the customer: *(i)* if the purchased product has been evaluated negatively, the Rate $R \in AK$ will assume a negative value by setting the ϕ coefficient to -1; *(ii)* if the merchant has been evaluated negatively, its identifier will be deleted from $PASet$ (w.r.t. G),

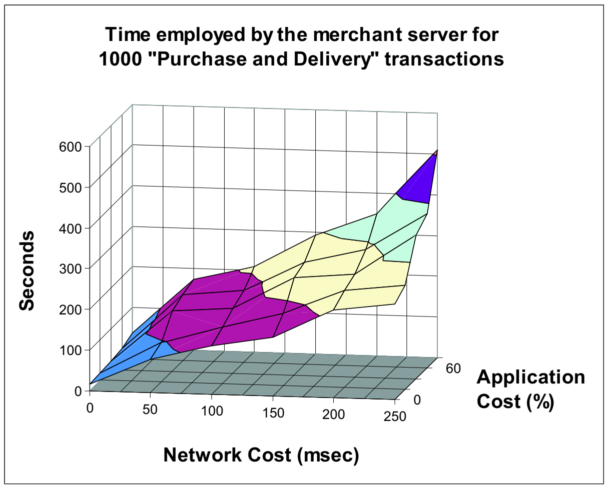


Fig. 4. The payment performances

then C might decide to inform M directly, or in an anonymous form by means of Ag, about her/his dissatisfaction.

V. System Prototype and Experiments

A complete prototype of MAST framework has been implemented in JADE [10], to test its CBB support activities simulating either single or continued sequences of CBB processes in a small B2C scenario[3]. Furthermore, to realize such experiments some EC sites have been realized in XML. In particular, in this section the results of the "Need Identification" and "Purchase and Delivery" stages are reported.

In the "Need Identification" the experiments have been finalized to measure the customer satisfaction degree (CSD) computed as the number of merchants' offers evaluated by the customers as correctly filtered by the system. The filtering process is carried out for each customer in two phases, the first performed by the Agency and finalized to disregard immediately all merchants' offers clearly out from the customer's interests and preferences; while the second phase is performed by the customer's agent to realize a fine tuning of the filtering activity on the basis of its profile. The tests have been carried out by 19 customers, using their agent profiles previously built on the basis of CBB activities carried out on XML EC sites.

More in detail, in the first phase on a total of 4750 merchant offers (250 offers for each customer), randomly chosen among the products offered in the XML EC sites, the agency has correctly rejected 3154 of them and considered potentially interesting 1596 offers. Then in the second phase, on the remaining 1596 offers the customers' agents have evaluated surely interesting only 193 offers, but 79 of them were not really interesting for the customers. In this way, we obtain a global CSD equal to 0,983. Note that we have set the filtering parameters to avoid the rejecting useful merchants offers.

About the "Purchase and Delivery" stage, it is clear that the cost located on the customers' client side has a minimal impact on the computational performance, since usually only one MAST activity runs at a time. Conversely, a merchant agent is associated with high computational cost, given that it has to satisfy a large volume of processes at the same time referring to different c agents. Consider that the server has to carry out other tasks for its EC activities that can absorb also a significant amount of resources and that have been simulated by assuming some different application costs as percentage of all processes carried out by an m agent; besides, the Internet cost can influence the global system performance and it has been simulated by setting some delays in the communications (tests were carried out on a 100 Mb LAN).

This procedure is surely a critical test activity, for the necessity to coordinate more parties, sending more messages and realizing some secure connections (SSL connections have been adopted in the tests). In the following, we employ this process only in order to obtain a rough estimation of the MAST computational efficiency in this CBB stage (keeping in mind that the MAST approach has anyway other significant benefits provided by the authentication mechanism and payment security level). In particular, the time necessary by the merchant's server to complete a sequence of 1000 "Purchase and Delivery" transactions for different application costs and network delays is represented in Fig. 4. Note that some steps occur using no secure connections and other occur on a simulated reserved banking channel.

The experiments suggest some considerations about the implemented MAST prototype and experimental results obtained. Using the MAST prototype, all CBB activities have been correctly carried out, customer and merchant profiles have been initialized, correctly updated and all our project goals have been meet, showing the capability of MAST to provide proper support to customers and merchants in EC scenarios. The experimental results obtained, even though they have only an indicative meaning both for the initial scenario assumption and some compulsory rough simulation show interesting performances in terms of efficiency, effectiveness and time employed.

VI. Related Work

The various aspects connected to the B2C have been dealt with in a very large variety of scientific works; some works which to our knowledge come closest to the material presented in this paper will be mentioned in this section.

The role of software agents in the EC has become very relevant, as proved by the large number of models and architectures proposed in literature and the state of the art has been investigated in a significant number of surveys [8], [9], [11], [12], [15], [19], [23]. The main opportunity offered by multi-agent systems is to support customers and merchants in performing their B2C activities. In the CBB context, MASs were traditionally focused on only in a few stages, usually "Merchant Brokering" and/or "Negotiation" stages, but progressively their support has been extended to all CBB stages (note that many MASs for B2C do not explicitly use the CBB model, but their activities are easily brought back to it). Furthermore, only a restricted number of MASs adopt one or more existent payment schemes explicitly, while the

[3]The simulations have been realized by employing computers based on single CPU (Intel Pentium 4, 3 GHz), RAM 2 Gbyte and O.S. Linux.

largest part of them just ignore the issue or record that a payment has occurred. Finally, since there is a large variety of protocols and communication languages that MASs adopt in B2C, these will not be specifically addressed here. In particular we propose the following three approaches:

- MAGMA [20] proposes a MAS for a free-market architecture based on messages. In MAGMA the agents are monitored by a central administration, only partially automatized; agents provide some trading strategies, independently by the users' behaviour, and can form agent alliances. Financial services for EC activities are provided, in a secure way, by a virtual bank that manages specific agents' account.
- CASBA [21], resulting from a CEE ESPRIT Project, implements an Internet agent-based marketplace supporting all CBB stages, in a flexible way with different auction types and dynamic negotiations, and some commercial payment schemas. CASBA employs Java, JavaScript, CORBA and XML technologies, while advertising is e-mail based. XML eases matching the data structures of the CASBA ontology with those of the client database.
- In [17] a model representing ontologies in a B2C scenario is proposed and a multi-agent architecture based on such model is described. It realizes a virtual marketplace agent-based where customers and merchants are supported by exploiting a representation of the concepts and behaviour involved in their EC world. Here, an agency has a central role as mediator in coalition formation, agent communications and in virtual auctions. No payment scheme is supported.

This aforementioned work exploit multi-agent systems to support B2C activities. They are explicitly CBB based or partially consistent with it. Customer interests and behaviour are taken into account in [17] where, similarly to CASBA, authors exploit XML to realize a unified representation in order to reduce the impact of heterogeneities. Payment issues are handled in MAGMA and in CASBA differently from MAST. Finally, MAGMA is designed to support also heterogeneous agents whereas CASBA, [17] and MAST deal only with homogeneous agents.

VII. Conclusion

This paper describes MAST, an XML-based multi-agent system to support customers and merchants in a suitable, homogeneous and personalized way, taking into account their interests on the basis of the behaviours shown during their B2C activities, represented as in CBB model. Furthermore, the opportunities offered by XML (for agent profiles, the messages, the inter-catalogue representation of products and categories and the agent communication language) are used along with those of a secure centralized payment scheme, based on existing financial institutions and single-use account numbers (payments happen only among financial institutions, over reserved communication channels, by preserving financial anonymity and confidentiality and benefiting of an existing authentication mechanism). Some results of experimental simulations in a small B2C scenario, carried out using a Jade-based prototypal implementation of MAST, are presented.

As for ongoing research, a development of MAST is planned by the introduction of different behavioural models taking in account emerging behaviours in the B2C area, such as formation of coalitions or the EC-site visiting.

References

[1] Asokan N., Janson P., Steiner M. and Waidner M. The State of the Art in Electronic Payment Systems. *IEEE Computer Magazine*, 30(9):28–35, September 1997.

[2] Financial Services Technology Consortium. Financial Agent Secure Transaction (FAST), Phase 1 Final Report (White Paper). FAST_Phase_1_White_Paper.pdf, September 2000.

[3] Foundation for Intelligent Physical Agents (FIPA). FIPA ACL Message Structure Specifi cation. SC00061G.pdf, December 2005.

[4] Freier A.O., Karlton P. and Kocher P.C. The SSL Protocol version 3. ssl-toc.html, November 1996.

[5] The Financial Services Technology Consortium (FSTC) website, 2006.

[6] Garruzzo S., Sarné G.M.L. and Palopoli L. AIPP: A FAST-Complied E-Payment Protocol. In *IADIS International Conference - Applied Computing 2006 (IADIS'06)*, pages 406–410, San Sebastian, Spain, April 2006. IADIS Press.

[7] Grosof B. and Labrou Y. An Approach to using XML and a Rule-based Content Language with an Agent Communication Language. In *Issues in Agent Communication*, number 1916 in Lecture Notes in Computer Science, pages 96–117. Springer-Verlag, Heildeber, Germany, 2000.

[8] Guttman R.H., Moukas A.G. and Maes P. Agent-Mediated Electronic Commerce: A Survey. *The Knowledge Engineering Review*, 13(2):147–159, 1998.

[9] He M., Jennings N. R. and Leung H. On Agent-mediated Electronic Commerce. *IEEE Transaction on Knowledge and Data Engineering*, 15(4):985–1003, 2003.

[10] The Java Agent DEvelopment (JADE) framework website, 2006.

[11] Jennings N.R. and Wooldrige M.J. Applying Agent Technology. *Int. Journal of Applied Artifi cial Intelligence*, 9(4):351–361, 1995.

[12] Liu J. and Ye Y. Introduction to E-Commerce Agents: Marketplace Solutions, Security Issues, and Supply and Demand. In *Proc. of the E-Commerce Agents: Marketplace Solutions, Security Issues and Supply and Demand*, volume 2033 of *Lecture Notes in Computer Science*, pages 1–6. Springer-Verlag, London, UK, 2003.

[13] The North America Industry Classifi cations (NAICS) website, 2006.

[14] O'Mahony D., Pierce M. and Tewari H. *Electronic Payment Systems for E-Commerce*. Artech House, Norwood, MA, 2nd edition, 2001.

[15] Palopoli L., Rosaci D. and Ursino D. Agent's roles in B2C E-commerce. *AI Communications*, 1912:95–126, 2006.

[16] Pinheiro R. Preventing Identity Theft Using Trusted Authenticatorss. *Journal of Economic Crime Management*, 2(1):1–16, 2004.

[17] Rosaci D. A model of agent ontologies for B2C E-Commerce. In *Proc. of the Sixth International Conference on Enterprise Information Systems (ICEIS'04)*, pages 3–9, Porto, Portugal, 2004.

[18] Shamir A. SecureClick: A Web Payment System with Disposable Credit Card Numbers. In *Proc. of the 5th International Conference on Financial Cryptography*, volume 2339 of *Lecture Notes in Computer Science*, pages 232–242. Springer-Verlag, London, UK, 2002.

[19] Sierra C. and Dignum F. Agent-mediated Electronic Commerce. Scientifi c and Technological Roadmap. In *Agent Mediated Electronic Commerce. The European AgentLink Perspective*, volume 1991 of *Lecture Notes in Artifi cial Intelligence*, pages 1–18. Springer, 2001.

[20] Tsvetovatyy M.B. and Gini M. Toward a Virtual Marketplace: Architectures and Strategies. In *Proc. of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'96)*, pages 597–613, London, United Kingdom, April 1996.

[21] Vetter M. and Pitsch S. Towards a Flexible Trading Process over the Internet. In *Agent Mediated Electronic Commerce, The European AgentLink Perspective*, volume 1991 of *Lecture Notes in Computer Science*, pages 148–162. Springer-Verlag, London, UK, 2001.

[22] W3C. Extensible Markup Language (XML) 1.1, February 2004.

[23] Ye Y., Liu J. and Moukas A.G. Agents in Electronic Commerce - Special Issue on Agents in Electronic Commerce. *Electronic Commerce Research*, 1(1-2):9–14, 2001.

J-ALINAs: A JADE-based Architecture for Linguistic Agents

Savino Sguera, Armando Stellato, Donato Griesi, Maria Teresa Pazienza
DISP, Università di Roma "Tor Vergata"
s.sguera@ieee.org, {stellato, griesi, pazienza}@info.uniroma2.it

Abstract—**In this paper we present J-ALINAs, a JADE-based Architecture for LINguistic Agents.**

The purpose of this architecture is to support communication between agents whose beliefs and intentions are driven by different, heterogeneous, knowledge models. This objective, often referred in literature as *semantic coordination*, can be carried on through the identification of specific agent roles and behaviors dedicated to the mediation of agents' knowledge.

In particular, such minimal hypotheses suggest an intelligent exploitation of natural language based technologies (resources and systems) as a necessary choice for capturing those similarities between the different knowledge models of agents trying to communicate, which are not in any way formally ratified.

We aim to provide a flexible framework to be adopted in open multi-agent environments across different scenarios, providing a further abstraction level from the underlying details related to specific semantic coordination approaches; a high-cohesion and low-coupling design, and an agent-interaction protocol make possible for the architecture to face non-*ideal* use cases optimizing the communications among the agents.

We discuss significant design issues, provide a prototypical implementation based on the JADE platform and a case study – MAPLE – integrating an ontology mapping component in the framework, showing flexibility of the architecture in real applications and its independence from any specific mapping algorithm.

Finally we will look at the semantic coordination protocol we designed from a strictly formal perspective, providing a CCS (Milner's *Calculus for Communicating Systems*) description of the protocol itself.

Index Terms—**Semantic coordination, ontology mapping, cooperating systems, intelligent agents.**

I. Introduction

Enabling communication between heterogeneous semantic peers (agents, semantic web services etc…) is a fundamental issue for the developing of the Semantic Web, especially in relation to the high levels of heterogeneity, evolution, distribution and autonomy of information which already characterize the Web as we know it now.

For the agents to be able to carry out the tasks they were designed for, they must impact the *communication barrier* between themselves: they will have to make their services available to the community, and recognize those of other *actors* in the community (to whichever extent it is considered, up to the Web as a whole); recognize, interpret and respond properly to communicative acts initiated by other agents and *understand* messages' content.

Although an important effort has been done by FIPA (Foundation for Intelligent Physical Agents) towards a standardization of agent platforms and agent message transport, too many *dimensions of variation* still are to be taken into account.

First of all, while adherence to the FIPA message protocol grants agents with the ability to establish a communication, distinguish between requests and responses, and bind them to a given subject of conversation, nothing is said about the real content of these messages, thus making impossible for agents belonging to different communities to actually communicate. We still have to cope with the heterogeneities of agent societies and, mostly, with the diversities in agent frameworks design which are often developed for ad-hoc environments and applications. All these aspects represent a threat against *large-scale* interaction among different multi-agent systems.

If peers were able to autonomously discover each other's services and to identify the specific problems which impede their communication, it would be much easier to devise solutions for supporting their communication, having a solid base to address the *impedance mismatch* among agents' different knowledge representations from a shared starting point.

II. Our Approach

In a scenario like the one depicted above, it appears evident that agents cannot rely on any shared form of understanding, their inner knowledge, as well as the functionalities they expose, being expressed and modeled upon ruling principles which are not known a-priori.

Our approach aims to:

- identify the fundamental *actors* and *roles* involved in the semantic coordination activity
- define agent-based mediation paradigms,
- *design* an agent-interaction protocol to support semantic coordination in a *flexible* fashion, and
- *model* the semantic coordination process giving the designed framework a high abstraction level from the

specific instance of the process itself.

Following these guidelines, we designed and developed J-ALINAs[1], a (JADE based) Architecture for LINguistic Agents: an agent framework for supporting semantic coordination between heterogeneous semantic peers. Main feature of this architecture is the abstraction from the *way* actually semantic coordination is carried out by a specific instance of an agent system, still considering it the main shared goal for the community. While abstraction from the techniques is maintained, on the contrary, it is important to identify the different kind of resources which may play a role in the process, in order to make them available when their contributions is required. In particular, we consider as necessary an intelligent exploitation of natural language (from which the name J-ALINAs arises) resources and processors, as language is the sole form of shared knowledge which is inherently adopted when expressing (through label descriptors, concept documentation etc..) the formal content of knowledge models and resources.

Moreover, we achieved a clear distinction between decision-making and service-providing competences, realizing a crucial separation between specific problem-solving skills and *strategic* knowledge about their composition to address a complex target, shared among the system's agents.

This allows to analyze new problems simply rearranging existing elements, and to improve or add functionalities without necessarily modifying the entire system.

III. Previous Works

A. State-of-the-art

A lot of work has been done by researchers towards flexible architectures to support semantic coordination in open multi-agent systems.

In [5] a three-layered peer-to-peer approach to ontology integration in a multi-agent system is described, to manage and deploy ontologies in a broad range of dynamic environments.

Within a multi-agent system however each agent expresses its own conceptualization, and in open systems interoperation between agents has to take into account their heterogeneous nature and background, that will likely take them not to fully understand each other, because of semantic misalignments which can easily arise.

A major requirement for agents interoperability [3] is for the semantic integration to be dynamic, that is, computed on-the-fly and not on the basis of pre-engineered mapping documents (which, even if considerable, may not always be available for any two given knowledge models). Agents shall have to be *self-describing*, and be able to characterize themselves, thus putting other agents in the condition of identifying the ones they need to cooperate with, in order to successfully communicate and exchange information towards a shared goal – that is in our case the agents' ontologies mediation.

To put it in Burstein's and Uschold's words [3], "*every agent must, in effect, wear its description 'on its sleeve'*".

Moreover, some interesting links between multi-agent systems and grid computing are given in [7], the formers complementing the latter for efficient services management without a-priori agreements, standing that multi-agent systems are groups of agents interacting and autonomously coordinating to satisfy a set of shared goals.

B. ALINAs: an open interacting-agent architecture

ALINAs [8], an Architecture for LINguistic Agents, developed by the Artificial Intelligence group of Tor Vergata, and former incarnation of the architecture which is presented in this paper, provided a set of three different classes of intelligent agents dedicated to supporting linguistic communication, each of which exhibits specific features related to the particular task it has been designed for:

- **resource agents**, which represent the beneficiaries of the communicative process, bearing some form of knowledge which need to be mediated against the one of other resource agents in the community
- **service agents**, providing support functionalities and holding responsibilities for some complex tasks occurring in the process
- **control agents**, having thorough knowledge of the problem to solve, control functionalities and decisional power in the agent society

This classification provides enough abstraction to ensure an incremental approach to systems development, whereas developers can focus on informative sources and resource agents and start to use available information before they design more advanced components in the system.

C. Linguistic Watermark and Semantic Coordination

As stated in [10], in order to make ontologies more prone to be mapped in distributed contexts, we believe it is necessary to revise the ontology development process to include, as a necessary part of this activity, the *enrichment* of ontology content with proper *lexical* expressions in natural language, such as synonyms and free natural language documentation, possibly in different languages.

Linguistic enrichment of ontologies (this is the name we gave to the process here described) requires a proper exploitation of several linguistic resources, which, due to the lack of defined standards for representation of linguistic knowledge, often differentiate upon many aspects, like structure, semantics, granularity of their content and representation. This strong heterogeneity led us to the development of the Linguistic Watermark [1], which is both a package providing generic abstract classes and interfaces for allowing uniform access to different linguistic resources, as well as a set of descriptors for identifying the characteristics of the accessed resources. These descriptors are important in our environment, as they allow agents to choose the linguistic resources (and thus contact the agents which grant access to their content) which are more appropriate for supporting a given communication.

Following the same intuition, it is also important, should an

[1] http://ai-nlp.info.uniroma2.it/software/J-ALINAs

ontology have been already linguistically enriched, to know in advance to which extent and through which modalities this enrichment process has been conducted. In our framework, this information is represented by the Ontological Linguistic Watermark [10], a collection of meta-data descriptors (see Figure 1) which expresses information about the (natural) language(s) adopted in describing ontology contents, and (eventually) about the linguistic resources which have been used to do that. These metadata play a central role in the semantic coordination phase we will describe extensively in section V.

```
<Linguistic Watermark> ::= { ontology enriched_by: <linguistic_resource> }

<linguistic_resource> ::=
              <linguisticResourceURI>,
              [ semantic_enrichment | linguistic_enrichment,<language> ]
              <enrichment_modality>
              <coverage>
<enrichment_modality> ::= [ supervised_enrichment | automatic_enrichment ]

<coverage> ::=  <conceptual_coverage>, <linguistic_coverage>
```

Figure 1: Specification of the Ontological Linguistic Watermark

D. MAPLE

MAPLE is a plug-in for the Ontology Editor Protégé [4], developed at the Artificial Intelligence research group of University of Rome, Tor Vergata, aimed to integrate ontology mapping facilities into the Protégé [4] Ontology Editing Suite.

It relies upon external linguistic resources – compliant to the Linguistic Watermark package – to obtain further information useful to find semantic correspondences between ontologies. The mapping process, before a deep inspection of the semantic characteristics of the two ontologies, preliminarily requires, at the linguistic level, to search for alternate expressions (synonyms) and/or glosses for the labels which describe ontology content, and also to identify proper translations in the more complex situation occurring when ontologies are labeled using different (natural) languages.

IV. J-ALINAS' ARCHITECTURE

A. The big picture

The generic meaning negotiation process between semantic peers usually has to start with a first hand-shake, in which information about the peers' respective knowledge is exchanged (in the form of ontology namespaces, as an example). Should the two agents find that they are not able to communicate due to any form of incompatibility between their form of knowledge, they will first try to coordinate in order to invoke the figures which can support their communication.

Suppose an agent wants to query a search engine, to obtain a link to a remote document; so far the scenario is particularly known (Figure 2). We then introduce this situation in the Semantic Web context, whereas documents will be expressed *against* a formal description of the content they describe: an ontology.

The agent, to effectively understand the document's content, will likely have the need to find mappings between its set of beliefs and conceptualization (its ontology), and the ontology the document is referred from.

To carry out the meaning negotiation and reach a knowledge model which is acceptable for both peers, the agent will request mapping service to a dedicated agent; the latter will take control of the whole negotiation process and will eventually need to request other agents' support or information regarding involved ontologies.

However, it is particularly interesting, again, to notice how this scenario is *independent* from the mapping techniques the dedicated coordinating agent will implement, whether they be based upon finding lexical anchors upon mapping documents or aimed to *spot* schema-level similarities, or to perform a combination of both.

The JADE[2] platform has been adopted to provide a prototypical implementation of the framework, in order to obtain a fully platform-independent prototype. JADE actually simplifies the development of agent platforms through a middleware complying to FIPA standards, implementing some of the required agents for the platform to be FIPA-compliant, and allowing agents to be movable from one machine to another.

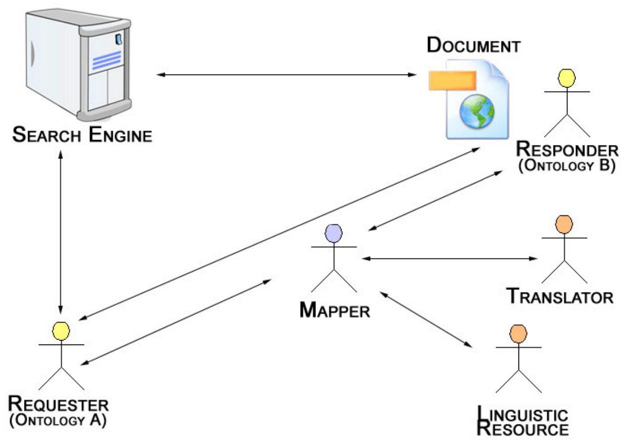


Figure 2: A generic application scenario

B. Agents and roles

J-ALINAs maintains much the same classification of agent roles which has been presented in section III.B, and more in details in [8] and [9]. Yet this approach is not to be intended as a rigid one: indeed the boundary between – for instance – service and resource agents will sometimes be not so clear, and will be possible for an agent to behave on both sides.

For this reason we believe it will be – in certain circumstances – more correct referring to resource, service or control *roles* rather than *agents*, since an agent could embody more than one role in the mediation process.

In other words it will not necessarily exist a *straight* 1-to-1 relation between an agent and the role it assumes in the

[2] http://jade.tilab.com

society, even though in the following paragraphs we will suppose it to exist, merely to have a distinct view of the identified competences. Here follows a more detailed description of the roles we have mentioned so far.

Linguistic agent

Linguistic agents encapsulate one or more linguistic resources, and provide an interface to access their content, through different kind of services.

The agent registers itself at a *Directory Facilitator* (DF) living in the agent platform, and publishes its services and the resources' URIs.

Ontological agent

An ontological agent is – together with the mapper agent – one of the fundamental actors of the process: usually it is such an agent to start the semantic coordination procedure and to request a conceptual mediation against another agent's ontology. He shall then assume the **requester** role (with this name we will refer to the ontological agent requesting the meaning negotiation), and choose the **responder** (the ontological agent holding the *destination* ontology); after that *he* will query the DF to search agents providing mediation services. We will discuss the **handshaking** phase in details in section V.

For the *semantic coordination* to take part in an effective way, each of the ontological agents willing to communicate publish a *fingerprint* of the conceptualizations they encapsulate (the Ontological Linguistic Watermark, OLW), containing information about the linguistic enrichment processes (as described in section III.C) which contributed to describe their knowledge content. As the ontology may have not been subject to any linguistic enrichment process at all, the OLW may even be reduced to the sole information about the idiom used to represent its concept identifiers. By comparing ontologies' watermarks, the mapper agent will then be able to easily decide which supporting agents are likely to be contacted, and negotiate the (natural) language(s) to be used throughout the whole mapping process.

Mapper agent

The only agent in the society to have the needed intelligence and a thorough knowledge of the problem, of the actors and of the methodologies to be applied to carry on a meaning negotiation activity, is the mapper agent.

We first need to distinguish between two different kinds of mapper agents: those which provide semantic coordination facilities by inspecting available ontology mapping documents, that is, resources available on the web which state conceptual correspondences between ontology resources, and those which offer the same service by computing those correspondences on-the-fly.

The first agent will mostly carry on basic look-up over available mapping documents (even more than one, as agent ontologies may have been built compositionally from several available smaller ones), supported by inferential abilities to entail new mappings other than those which are explicitly declared.

For those cases where mapping documents are not available (or they do not cover completely the addressed conceptualizations) – presumably not a *small* fraction of *real world* situations – the second kind of mapping agent is invoked to try to individuate and establish possible semantic similarities by inspecting the structure of the two ontologies and by exploiting information provided by linguistic agents.

C. The agents' behaviors

We designed a set of JADE behaviors, realizing the system's reactive layers hierarchy. In the following paragraphs we will describe the most significant ones, *moving* our perspective from one agent to another.

1) Linguistic support to ontology mapping

The behavior encapsulating access to linguistic resources and providing linguistic support to the ontology mapping process is a cyclic one, enabling the linguistic agent to react to mapper agent's stimuli. In other words it models a simple reactive behavior, implementing a reactive agent paradigm.

A linguistic agent in the *setup* procedure will add an instance of such behavior to its execution queue; it will eventually reply to queries originated by the mapper agent with semantic anchors for the particular term, or with a *NOT_UNDERSTOOD* FIPA message performative in such cases where the query is malformed (or, more simply, out of its comprehension).

Queries are matched by the agent against an application-defined message template, providing developers with high freedom of choice with respect to the type and number of linguistic services the agent publishes in the system.

The *linguistic-querying* task is delegated to a specific behavior residing in the mapper agent (Figure 3), which is responsible for persisting anchors upon the agent itself; however the concept of *persisting* anchors is absolutely abstract for the framework, and it is responsibility of the application developer to implement the *persist* and *retrieval* functions.

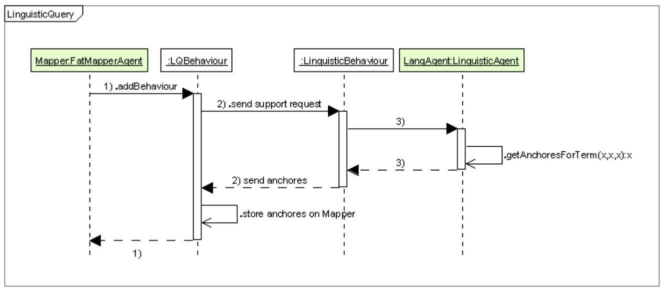


Figure 3: Interaction between mapper agent and linguistic resources

2) The twofold ontological agent behaviour

The provided ontological agent paradigm has been designed to act indifferently as *requester* or *responder* in the mapping process. We actually reckon this has positive implications whether towards bidirectional knowledge mediation between agents, and the integration of the application-specific ontological agent into the surrounding environment, reducing the number of different *actors* for the developers to take into account.

If the agent is required to act as *requester* in the process it will start discovering the surrounding environment, contacting agents and *handshaking* them.

Otherwise, if the stimulus is a *PROPOSE* FIPA message performative, the agent will eventually accept to participate in a mediation and wait for a mapper agent to ask for its ontological linguistic watermark or information about its ontology's concepts and relations among them.

3) *The ontological agent's introspection*

Concurrently, the ontological agent runs an instance of a *simple reflex* behaviour aimed to answer specific queries about ontology's instances, properties or relations among concepts. This behaviour can be exploited by developers to fit messages exchange among agents to support the specific mapping algorithms, and does not affect the handshaking procedure at all.

4) *The generic ontology mapping process' instance*

Modules providing ontology mediation based upon external linguistic resources have been designed to give the handshaking procedure a *well-known* form, and at the same time to give developers using the framework the chance to easily *extend* the system with custom behaviours encapsulating application-specific mapping algorithms.

Indeed, after the *handshaking* phase took place the mapper agent shall own all the elements needed to conduct a *generic* ontology mediation, and to effectively decide which other agents in the society are likely to be involved in the process; in other words we provide developers embracing J-ALINAs architecture a solid and shared base to design their application on, building a further abstraction level from the underlying environment.

The abstraction itself comes to evidence even from a strictly *sequential* perspective: because every mapping process starts with a communication among agents and the system's facilitator whereas the agents discovery the surrounding environment, we reckon it is convenient to let developers the chance not to care – to some extent – about the *coordination* and *synchronization* layer, focusing on their system's peculiar aspects.

V. SEMANTIC COORDINATION: THE AGENT-INTERACTION PROTOCOL

The interaction protocol for the semantic coordination is intended to be the whole project's *keystone*, designed to adapt itself to different situations, minimizing the number of messages the agents must exchange – as it is normal to expect in a distributed environment – to achieve their communication's main purpose: the mapping of agents' ontologies (or, more specifically, of the concepts they need to express in their communication).

Throughout the protocol requisite elicitation and analysis phases, the chance for an agent to *embody* more than one role in the mediation process initially emerged as a problem: indeed, this implies the agent to be *aware* of its manifold functions while interacting, discovering and choosing which other agent to query. For instance, an agent implementing both the mapper and the ontological paradigm could prefer to rely upon its own mapping *skills* instead of contacting a third-party.

This brought the need for the protocol to adapt itself to variations in the system's *topology* (Figure 4) and in the message flow among agents, and at the same time to ensure developers freedom of choice in deploying functionalities on application-specific agents.

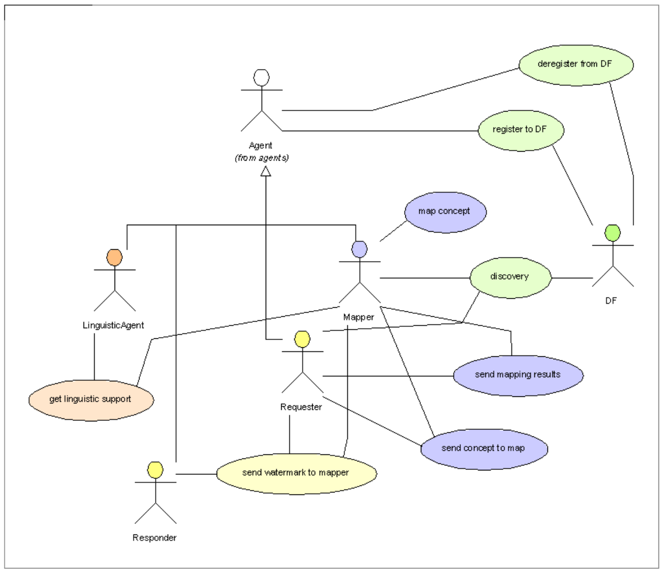


Figure 4: J-ALINAs use case diagram

After the setup phase, the agents are in a stand-by state. Whenever the *requester* receives the stimulus to initiate the process, an *handshaking* activity (Figure 5) introduces the communication. The *DF* will be queried to discover which other ontological agents are living on the platform. The *requester* agent will then elaborate the search results, and choose the passive agent (*responder*) in respect to fully application-defined criteria.

The mapping proposal is sent to the chosen *responder* agent, who will eventually accept the proposal, notifying the *requester* with an *ACCEPT_PROPOSAL* FIPA message performative including its ontology URI.

The *requester* will then query the *DF* asking for the presence of formal-model based mapper agents providing mappings between its ontology and the *requester*'s one.

Whether such an agent exists in the platform, the *handshaking* phase stops here, and comes the time for the *requester* to start querying the mapper with concept-mapping requests.

In the other case, another query is submitted to the *DF*, this time looking for the existence of a mapper agent based upon external linguistic resources exploitation. If the query is successful, the *requester* shall choose which mapper agent to contact and send out a *PROPOSE* message performative, waiting for the eventual acceptance from the counterpart. The proposal message will include the agent's ontological linguistic watermark and the *responder* unique identifier.

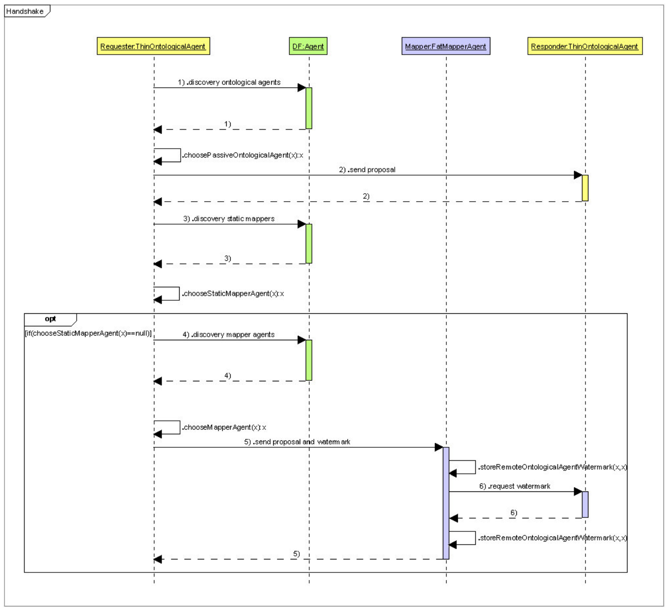

Handshake
Requester:ThinOntologicalAgent
DF:Agent
Mapper:FatMapperAgent
Responder:ThinOntologicalAgent
1) .discovery ontological agents
1)
.chooseP assiveOntologicalAgent(x):x
2) .send proposal
2)
3) .discovery static mappers
3)
.chooseStaticMapperAgent(x):x
opt
[if(chooseStaticMapperAgent(x)==null)]
4) .discovery mapper agents
4)
.chooseM apperAgent(x):x
5) .send proposal and waterm ark
.storeRemoteOntologicalAgentWatermark(x,x)
6) .request watermark
6)
.storeRemoteOntologicalAgentWatermark(x,x)
5)

Figure 5: Handshaking - sequence diagram

The semantic coordination is now complete: the mapper agent has all the information needed to negotiate the natural language to use throughout the mediation, and choose which other agents to contact; the actual ontology mapping process can now start. Whilst no other agent was holding control of each others' actions and of message flow in the system so far, now the decisional power goes completely to the mapper agent.

The adoption of such a protocol does not violate the dynamics of the process and the heterogeneous nature of actors which are typically associated to the idea of Semantic Coordination. On the mere perspective of ontological agents (which could be considered as the *end users* of Semantic Coordination), they just need to be *aware* of the existence of Mapping Agents willing to help them in the process of communicating with other agents based on different kind of knowledge, and implement (as they are expected to do in whatsoever scenario) basic speech acts for requesting their help. Knowledge of an ontological agent's own linguistic expressivity is also requested in our paradigm, but this in line with recent trends in the Semantic Web area, where ontologies need to be expressed in a linguistically motivated fashion [2,7], possibly considering integration with existing linguistic resources [1]. The core of the mapping process is then delegated to service agents and resource agents, thus not requiring any strict protocol to be followed by ontological agents.

VI. A Case Study: MAPLE

As anticipated in III.D, MAPLE is a Protégé plug-in integrating ontology mapping functionalities in the Protégé ontology editing tool. MAPLE also provides ontology mapping as a standalone process, though it is not conceived for a distributed environment.

We succeeded in abstracting MAPLE's algorithms and their relations with linguistic resources from its original scope, and provided an implementation for a mapping agent which *fits* in a distributed environment according to the described framework.

From MAPLE's algorithms' analysis what soon emerged is a linear relation between the *cardinality* of the *target* ontology and the number of *queries* the mapper agent is supposed to submit to linguistic resources (especially for the resource which is expressed in the natural language adopted for the communication). Standing on MAPLE mapping algorithms, not all the queries can be estimated in advance and sent as a unique request to a linguistic agent, thus requiring an heavy communication load between the mapper and the linguistic agents. This makes unacceptable the approach with autonomous linguistic agents to hold information sources, because the *size* of the involved ontologies is obviously not estimable in advance.

So we looked for a trade-off between the mapping process' distribution and the communicational complexity among agents: this resulted in one single agent implementing both the mapper and the linguistic agent paradigms. This way we kept constant and independent from ontologies' size the number of messages to be exchanged in the system, encapsulating in a single agent – thus in a single physical host – mapping algorithms and linguistic resources accesses.

The eventual need for a translation phase can easily be delegated to an external agent: the label(s) associated to the concept to map can be sent to a linguistic agent providing the translation services which will then send back to the mapped the translated terms; the mapper will then try to individuate the right translations among those provide by the translator, and then start to work in the language it is specialized in. This particular framework individuates a specific figure for mediation, given by a mapper agent which is particularly proficient in one (or more) idiom and which must be contacted only when its idiom pertains to at least one of the communicating ontological agents' OLWs.

VII. Conclusions

One of the most interesting perspective to look at JADE – and obviously at JADE-based systems – is the possibility to easily design – through extremely interesting extensions such as JADE-LEAP (Light Extensible Agent Platform) – agents living in mobile platform and devices.

Looking at the Semantic Web evolution together with the proliferation of wireless devices – PDAs and mobile phones – it becomes quite clear what the web is likely to become in the next years: a *dense* and highly dynamic network, populated by a multitude of *nomadic* elaboration nodes able to *understand* information sense, communicating in a more effective and *intelligent* way.

We believe it is natural – to some extent – to look at these *nodes* as autonomous entities communicating in a peer-to-peer fashion, *living* in an heterogeneous environment, providing and asking services one to each other; that is, actually, as agents.

As we already discussed, semantic coordination will be an essential and binding part of this communicative process: when most of the information will be expressed with respect to these specifications; knowledge mediation will be a very frequent *runtime* process, providing the base for mutual comprehension between systems.

Also, we believe a distributed approach has to be considered fundamental. This is one of the most interesting perspective to look at our work, which is actually providing a *flexible* framework for multi-agent systems development in response to the different needs coming from several backgrounds and practical applications.

We took into account the communicational effectiveness and efficiency, providing the framework with enough *granularity* to let developers easily control and keep *constant* the number of messages exchanged by the agents living in the particular instance of the system, leaving open the road to developments in scenarios with very specific constraints like those expressed in the *mobile computing* area.

APPENDIX

The handshaking procedure: a formal review

In this paragraph we will give a – simple – formal description of the handshaking phase and the coordination protocol we designed, through Robin Milner's *Calculus for Communicating Systems* (CCS), [6]. More specifically, what we will do is looking at each agent of the society as an independent process, and model its behaviour in terms of others'.

We first define the generic *Directory Facilitator* as a recursive process:

$$DF = (search + searchFederated) . \overline{result} . DF$$

We then define the twofold ontological agent behaviour as follows:

$$OA \xrightarrow{Req} OA_1$$
$$OA \xrightarrow{Prop} OA_2$$

where *Req* stands for the receiving of a REQUEST FIPA performative, and *Prop* for the receiving of a PROPOSAL FIPA performative, and specify that:

$$OA_1 = (\overline{search} .result.chooseP. \overline{proposeP} . \overline{search} .result).$$
$$(chooseFM + \overline{search} . chooseM . \overline{proposeWM})$$

where *chooseP* and *proposeP* indicate respectively the actions of choosing the passive ontological agent and notifying it with the proposal to collaborate in the mediation process, and *chooseFM* and *chooseM* indicate the action of choosing the Formal Mapper agent, where available, and

$$OA_2 = proposeP . \overline{watermark} .$$

The mapper agent will behave as follows:

$$MA = proposeWM . storeWM . watermark . storeWM$$

where *WM* stands for Watermark

Now we can define the whole *handshaking* process as processes above running concurrently and stimulating each other, limiting each process' interface to allowed communication channels:

$$Handshake = (OA_1 \mid DF \mid MA \mid OA_2) \backslash chooseP \backslash chooseFM \backslash chooseM \backslash storeWM. \backslash searchFederated$$

REFERENCES

[1] Benjamins, V. R.; Contreras, J.; Corcho, O.; and Gómez-Pérez, A. 2002. Six Challenges for the Semantic Web. In Proceedings of SemWeb@KR2002 Workshop, 19-20 April 2002, Toulose.

[2] P. Buitelaar, T. Declerck, A. Frank, S. Racioppa, M. Kiesel, M. Sintek, R. Engel, M. Romanelli, D. Sonntag, B. Loos, V. Micelli, R. Porzel, P. Cimiano. LingInfo: Design and Applications of a Model for the Integration of Linguistic Information in Ontologies In Proceedings of the OntoLex Workshop at LREC, pp. 28-32. May 2006.

[3] M. Burstein, M. Uschold: "Infrastructure for Semantic Interoperability and Integration: breakout discussion summary", Dagstuhl Seminar Proceedings: Semantic Interoperability and Integration 2005

[4] Gennari, J., Musen, M., Fergerson, R., Grosso, W., Crubézy, M., Eriksson, H., Noy, N., and Tu, S.: The evolution of Protégé-2000: An environment for knowledge-based systems development. International Journal of Human-Computer Studies, 58(1):89–123, 2003.

[5] L. Li, B. Wu, Y. Yang: "Semantic mapping with multi-agent systems", 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE'05)

[6] R. Milner: "Communication and Concurrency", Prentice-Hall International, Englewood Cliffs, 1989.

[7] A. Oltramari, C. Huang, A. Lenci, P. Buitelaar, C. Fellbaum. OntoLex2006: Second Workshop on Interfacing Ontologies and Lexical Resources for Semantic Web Technologies, held jointly with LREC2006, Magazzini del Cotone Conference Center, Genoa, Italy

[8] M. T. Pazienza, A. Stellato, M. Vindigni: "ALINAs: un'architettura multi-layer ad agenti per il supporto alla comunicazione linguistica", WOA2002, Milano (Italy), 18 November 2002

[9] M. T. Pazienza, M. Vindigni: "Agent based ontological mediation in IE systems", LNAI 2700, Summer Convention on Information Extraction 2002, Information Extraction in the Web Era, pp. 92-128

[10] M. T. Pazienza, A. Stellato: "Linguistically motivated Ontology Mapping for the Semantic Web", SWAP 2005, the 2nd Italian Semantic Web Workshop Trento, Italy, December 14-16, 2005

[11] M. T. Pazienza, A. Stellato An Environment for Semi-automatic Annotation of Ontological Knowledge with Linguistic Content 3rd European Semantic Web Conference (ESWC 2006) Budva, Montenegro, June 11-14, 2006

Software Agents for Autonomous Robots: the Eurobot 2006 Experience

Vincenzo Nicosia[1], Concetto Spampinato[1] and Corrado Santoro[2] for the Eurobot DIIT Team
Università di Catania
[1]Facoltà di Ingegneria - Dipartimento di Ingegneria Informatica e delle Telecomunicazioni
[2]Facoltà di Informatica - Dipartimento di Matematica e Informatica
Viale A. Doria, 6 - 95125, Catania, Italy

Abstract— **Agent-based software architectures have been used and exploited in many application fields. In this paper, we report our experience about using intelligent agents for an unusual task: controlling an autonomous robot playing a kind of "golf" game in an international robotic competition. Driving a real robot is a practical application field for software agents, because different subsystems need to be controlled and synchronised in order to realize a global game strategy: cooperating agents can easily fit the target. Since this application requires a soft real-time platform to guarantee fast and reliable actions, and also a valuable communication system to gain feedback from sensors and to issue commands to actuators, we chose Erlang as programming language. A two-layer multi-agent system was thus designed and realized, composed of a lower layer, hosting agents taking care of the interface with sensors and actuators, and a higher layer, where agents are in charge of "intelligent" activities related to game strategy.**

Keywords— Mobile and Autonomous Robots, Computer Vision, Autonomous Agents, Real-Time Systems, Erlang.

I. Introduction

Software agents are autonomous entities that, living in a virtual world, are in charge of accomplishing the goal they are programmed for. In doing so, agents interact with the environment where they live in, by sensing its state and acting onto it, in order to achieve their goal. For these reasons, they are often called "software robots".

In spite of this similarity between (software) agents and (real) robots, agents, and above all multi-agent systems, are mainly exploited in realizing complex software systems and applications requiring intelligence, flexibility, interoperability, etc., while the area of robotics is often a matter of research on real-time and control systems. However, when a (autonomous) robot needs some intelligence to perform its activities in a more efficient and effective manner, the use of agent technology seems a natural choice [17].

The issue is that, in these cases, agents have to face the problems related to the interface to physical sensors and actuators, which connect the computer system with a physical environment that also changes during time. Therefore, an agent–enabled robot has not only to tackle the problems related to direct use of input/output ports, acquisition and driving boards, serial ports etc., but it should also take in account the fact that the scenario is time-constrained. In fact, as it is known, an information acquired from sensors (e.g. the position of the robot or of its arm) has a deadline after which the data become stale and no more useful, unless a fresh value is obtained. These problems are quite known in the area of real-time systems and their solution is achieved by means of platforms and/or operating systems that regulate program execution—in terms of process/task scheduling, race condition and delay control—in order to guarantee that deadlines are met.

Since such a real-time support is needed also in the case of the use of an agent-based system to control robot activities, the traditional and well-known agent platforms, which are mainly based on Java, cannot be employed at all: at it is known, the main problem of Java is the garbage collector, which introduces unpredictable latencies that prevent any attempt to build a time-constrained system. Indeed, RTSJ specification [6] provides a set of classes and some programming rules that allow the realization of real-time Java systems, but the specification introduces hard constraints in object allocation and reference that require an existing Java program (and thus an agent platform) to be rewritten in order to make it RTSJ-compliant [22], [16], [8].

In the context of agents and real-time systems, a language that features some interesting characteristics is *Erlang* [5], [4], [1]. It is a functional and symbolic programming language that has been proved to be suitable for the implementation of multi-agent and intelligent systems [21], [10], [12], [11], [13], [15], [14], [9]; moreover, since the Erlang runtime system is able to provide *soft real-time*[1] capabilities [18], [3], it seems also quite useful for the realization of an autonomous robot controlled by autonomous agents. In this context, this paper describes the authors' experience in designing and implementing an autonomous robot, for the Eurobot 2006 competition[2,3], by means of a multi-agent system written using the Erlang programming language. A layered multi-agent system has been designed, composed of two layers: a *back-end* (lower layer), comprising agents performing the interface with robot's physical sensors and actuators, and handling low-level control activities; and a *front-end* (upper layer), hosting agents dealing with the game strategy. Thanks to this layered architecture, hardware-level interactions and

[1]A system is called *soft real-time* if it is able to take into account deadlines, but if a deadline is not met, it has no particular consequences [19], [20].

[2]`http://www.eurobot.org`

[3]`http://pciso.diit.unict.it/~eurobot`

intelligent activities are clearly decoupled, making the design and implementation of the software system more easy, and also allowing the programmer to easily reuse some parts and/or to improve or change the functionalities of the system.

The paper is structured as follows. Section II describes the game that robots have to play at Eurobot 2006. Section III illustrates the basic hardware and mechanical structure of the robot developed. Section IV deals with the software architecture of the control system of the robot, describing the agents composing the system, their role and their activities. Section V discusses some implementation issues. Section VI reports our conclusions.

II. The Game at Eurobot 2006

Eurobot is an international robotics competition which involves students and amateurs in challenging and amazing robot games. The main target of the event is to encourage sharing of technical knowledge and creativity among students and young people from Europe and, in the last two editions, from all around the world.

Every year a different robotic game is chosen, so that all teams start from the same initial status and new teams are stimulated to participate. Here we report an overview of the rules for the 2006 edition of Eurobot[4], when the selected game was “Funny Golf”, a simplified version of a golf game where robots had to search balls in the play-field and to put them into holes of a predefined colour.

A. Field and Game Concepts

As Figure 1 shows, the play-field is a green rectangle of 210x300 mm, surrounded by a wooden border. Borders on the short sides of the field have a red (resp. blue) central stripe which delimits the starting area for each robot. The field has 28 holes, 14 of them encircled by red rings and the other by blue rings. A total amount of 31 white balls and 10 black balls are available during the game. Fifteen white balls and two black balls are placed into the playing area at predefined positions, while four more black balls are randomly positioned into holes, two for each colour. The remaining balls (sixteen white and four black) could be released by automatic ejection mechanisms positioned at each corner of the field. Finally, four yellow “totems” are positioned into the field and are both obstacles for robots and switches for the ball–ejection mechanisms.

Robots must be absolutely autonomous: any kind of communication with the robot, both wired or wireless, is not allowed during matches. Robots have spatial limits, in terms of height, perimeter and so on, and have to pass a homologation test before being accepted for the competition. Each robot can also use any kind of positioning and obstacle-avoidance system, and supports are provided at the borders of the playing area to place (homologated) beacons, if needed.

[4]This edition took place in Catania, Italy.

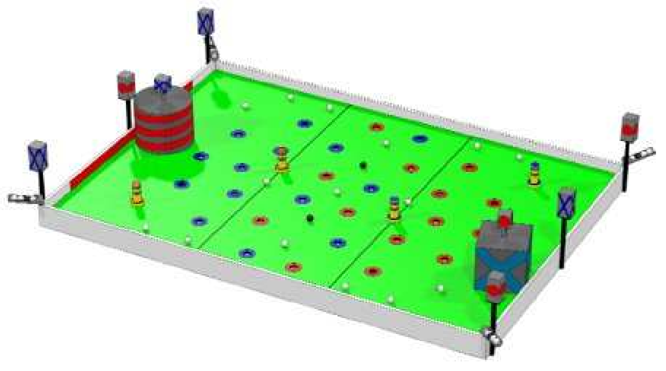


Fig. 1. The playing area

B. Playing Funny Golf

Before starting, each robot is assigned a colour, either red or blue. Robots start from the border opposite to their playing area, i.e. in the opponent’s field, and at least one side of the robot must touch the starting area (short border of the play-field). After robots are placed into the field and all setup procedures by team members are over, the referees choose the positions of totems and black balls, by means of a random selection. When all the components in the play-field are set up, one of the referees gives the start signal and robots can play. Each robot has to put as many white balls as possible into its holes in a time of 90 seconds. Robots can also put black balls into opponent’s holes, suck them out of their holes, or even suck white balls out of opponent’s holes. There is no restriction about strategies or techniques adopted in order to search, catch, release and suck out balls. It is not allowed to hurt the other robot or to obstacle or damage it in any way. It is neither permitted to damage the playing area or playing objects (such as balls, holes, totems or ejecting mechanisms). The ejecting mechanisms can be triggered by touching a totem for a given amount of time: this closes a simple electric circuit and allow balls into the ejector to be released. At the end of the match, each white ball in the right hole is considered as a point, and the robots which has the highest score is the winner.

III. The DIIT Team Robot

Building an autonomous robot to play “Funny Golf” is not a trivial task, since different subsystems are needed to perform ball searching, catching and putting, and many physical constraints are imposed by game rules themselves. The following subsections describe the robot realized by the DIIT Team[5], which participates (for the first time) to the 2006 Eurobot edition.

A. The Core

An embedded VIA 900Mhz CPU is the core of the robot. We used a motherboard produced by AXIOM Inc. which incorporates Ethernet, parallel port, 4 serial ports, USB, IDE

[5]“DIIT” means Dipartimento di Ingegneria Informatica e delle Telecomunicazioni.

controller and other amenities (such as PC/104 bus, not used in our configuration). The operating system used is a Debian GNU/Linux (Etch), with kernel 2.6.12 and glibc 2.3.5. GNU/Linux was selected because of its stability and robustness, that are important features when driving a robot.

B. Locomotion System

In order to guarantee fast movements, we decided to use a locomotion system based on two independent double-wheels, driven by DC motors. Wheels diameter is small enough to allow fast rotation and large enough to avoid holes. DC motors are directly connected to a motor-controller, driven by a RS232 serial line. The controller allows to set different speeds for each wheel, both for forward and backward directions. Each wheel is connected to an optical encoder, driven by a serial mouse circuitry, which feeds back to the software system information about real rotation speed and position of the wheel. This information is then used by the Motion Control agent to adjust the speed and the trajectory.

C. Vision

Searching balls in the playing area requires a kind of vision system to find them. We chose to use a simple USB webcam to capture video frames at a rate of about 4 frames/sec, still enough to guarantee an accurate and fast analysis of objects in the field. The webcam is able to “view” the field from 30 to 160 centimetres in front of the robot, with a visual angle of about 100 degrees in total. Frame grabbed by the webcam are passed to the “Object Detector” agent, which filters them to find balls (both black and white) and holes (both red and blue).

D. Catching and putting balls

Once balls are detected, it is necessary to put them, somehow, into the right hole. We decided to suck balls using a fan, and to choose where to put them using a simple selector, driven by a servo-motor. Balls are saved into a small buffer if they are white and the buffer has enough space, or ejected out if they are black or if the buffer is full. The fan is powerful enough to suck balls at a distance of about 12 centimetres from the front side of the robot, and it is also able to suck balls out of holes when a special small bulkhead on the front side is closed. A simple release mechanism, which uses a servo-motor, allows balls to be dropped down to the final piece of the buffer and to fall into a hole.

E. Sensors and Positioning

Many sensors have been used onto the robot. First of all, a colour sensor for balls is installed into the ball selector, to recognise if a sucked ball is white or black. A complex system of proximity sensors is installed in the bottom side of the robot to recognise holes when the robot walks over them, and to allow a smart and fine positioning during the ball putting phase. A presence sensor (made by a simple LED–photo-resistor couple) is placed in the final part of the buffer, to reveal the presence of a ball ready to be dropped into a hole. The same sensor is used to detect when the ball has been successfully put into a hole.

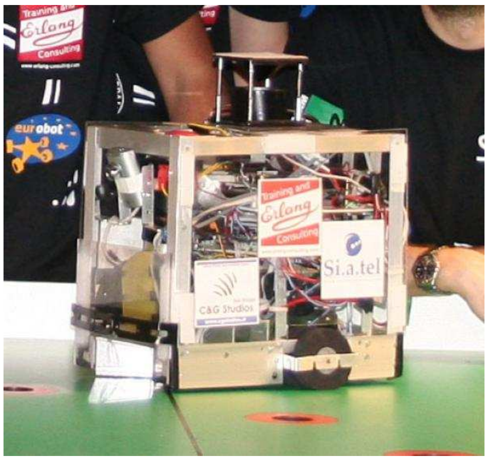


Fig. 2. The Robot in the playing area

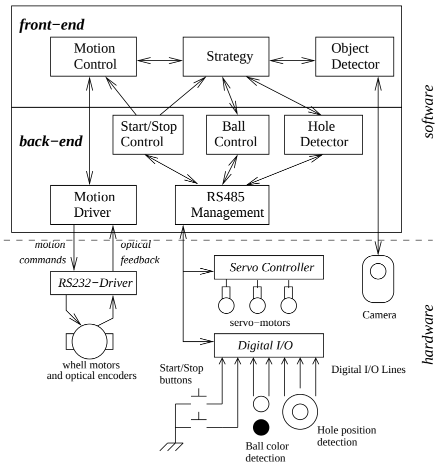


Fig. 3. Hardware/Software Architecture

IV. The Robot’s Software Architecture

Given the robot structure illustrated in the previous Section, it is clear that the implementation of the system to control it has to face some problems that are not present in traditional (only software) multi-agent systems: the interface with physical sensors and actuators. For this reason, the basic software architecture of the robot, which is sketched in Figure 3, is composed of *two layers*, (*i*) a lower one, called the *back-end*, including reactive-only agents, responsible for a direct interaction with the hardware, and (*ii*) a higher layer, called the *front-end*, hosting the “robot’s intelligence” by means of a set of agents implementing the artificial vision system, the game strategy, the motion control, etc., and interacting with back-end’s agents in order to sense and act onto the environment. All of these agents comply with an ad-hoc model which, together with the details on functionality of the overall system, is described in the following Subsections.

A. Agent Model

As reported in Section I, due to real time requirements and other peculiarities of a robotic application, well-know Java-based agent platforms cannot be employed; therefore, according to authors' past research work [21], [10], [12], [11], [13], [15], [14], [9], we decided to use the Erlang language [5], [4], [1] for the development of the robot's software system. In addition to its soft-real time features, Erlang has a concurrent and distributed programming model that perfectly fit the model of multi-agent systems: an Erlang application is in fact composed by a set of *independent processes*, each having a *state*, *sharing nothing* with other processes and communicating only by means of *message passing*. Such processes can be all local (i.e. in the same PC) or spread over a computer network; this is transparent to the application because the language constructs for sending and receiving messages do not change should the interacting processes be local or remote.

Given these features and the requirements for the robot control application, a suited agent model has been developed, which is based on two abstractions called *BasicFSM* and *PeriodicFSM*. The former, *BasicFSM*, is essentially a finite-state machine model, in which transitions are triggered by either the arrival of a message or the elapsing of a given timeout, and a specified per-state activity is executed (one-shot) when a new state is reached. The latter, *PeriodicFSM*, is instead a finite-state machine in which transitions are activated only by the arrival of a message, while the per-state activity is executed, when a state is reached, *periodically*, according to a fixed time period and within a deadline, which is equal to the period itself.

As it will be illustrated in the following, BasicFSM model is used for front-end agents, while the PeriodicFSM model is essentially exploited for those interacting with sensors and actuators and thus running in the back-end.

B. The Back-End

As Figure 3 illustrates, the *back-end* layer is composed by the following agents: *Motion Driver*, *RS485 Management*, *Start/Stop Control*, *Ball Control* and *Hole Detector*. All of these agents use the PeriodicFSM model but only the first two are directly connected with hardware resources.

The *Motion Driver* agent is in charge of driving wheel motors and gathering feedback from optical encoders. It basically handles messages (sent by front-end agents) specifying the *speed* to set for the left and right wheel, forwarding it (after measurement unit conversion) to the motor controller connected through the RS232 line. On the other hand, its periodic activity entails receiving the feedback from optical encoders (i.e. tick count), acquired through another RS232 line, and then computing tick frequency, thus evaluating the real speed of the wheels: the obtained value is used to adjust the value(s) sent to motor controller in order to make each wheel to reach the desired speed[6].

The *RS485 Management* agent is responsible for driving two external boards connected, to the PC, through the same RS485 serial bus: a controller for servo-motors and a board offering a certain number of I/O digital lines. Since each servo-motor and each I/O line is then used by different agents, the RS485 Management acts as a de-/multiplexer for actions and sensed data. Its periodic activity is the sampling of digital inputs, by means of a request/reply transaction through serial messages exchanged with the I/O board; polled data are thus stored in the agent's state in order to make them available for requests coming from other agents. In addition, the RS485 Management is able to receive messages containing commands to be sent to servo-motor, through the servo-controller; in particular, each command specifies the servo-motor to drive and the rotation angle to be set.

The *Start/Stop Control* agent is a reactive one that periodically queries the RS485 Management in order to check if the "start" or "stop" buttons have been pushed. On this basis, it sends appropriate start/stop messages to the Strategy agent (see below) in order activate (resp. block) its behaviour when a match begins (resp. ends). Since the duration of a match is fixed (90 seconds), this agent embeds also a timer that, armed after a start, automatically sends a stop message when the 90 seconds are due.

The *Ball Control* agent is responsible for managing the ball sucking system, the buffer and the ball release system. During its periodic activity, it queries the RS485 Management agent in order to check the input lines signalling that a new ball has been sucked: if this event occurs, on the basis of the colour of the ball[7], it drives the sucking system's arm servo-motor in order to put the ball in the buffer—if the ball is white and the buffer is not full—or to throw the ball away—if the ball is black or the buffer is full. This agent also holds the number of balls in the buffer, information that, queried by the Strategy agent, is used by the latter to control robot behaviour. As for ball release, the Ball Control agent, following a proper command message, is able to interact with the RS485 Management agent and thus drive the servo-motor controlling the release of a ball. Finally, by checking the status of another input digital line, the Ball Control agent is able to understand if a released ball has been successfully put into a hole.

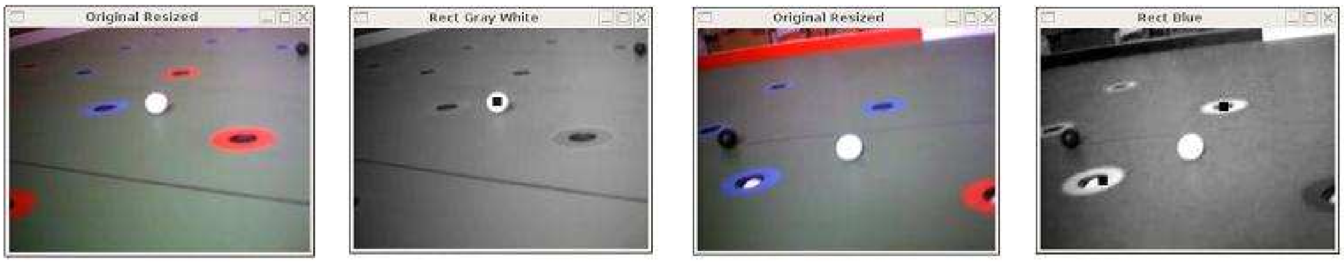
The last agent of the back-end, the *Hole Detector*, reads, through a proper interaction with the RS485 Management agent, the data coming from proximity sensors placed under the robot for hole detection and positioning. It is able to understand the position of the robot, with respect to the hole to catch, and can thus forward this information to the Strategy agent, which, in turn, will drive the wheels to centre the hole and put the ball into it.

C. The Front-End

The front-end layer implements the high-level activities that drive the robot to reach its goal, i.e. placing the most quantity

[6]This is obtained by means of a proportional-integrative-derivative software controller.

[7]The colour is detected through a sensor connected to another digital input line.



(a) (b) (c) (d)

Fig. 4. Recognition by Object Detector agent

of balls into its holes. This layer is composed of three agents: *Object Detector*, *Motion Control* and *Strategy*.

The *Object Detector* has the task of observing the playing area, by means of a USB camera, detecting the objects needed for the game, i.e. balls and holes, and computing their coordinates with respect to the robot position. Since it uses a computation-intensive image manipulation algorithm, this is the sole agent written in C and not in Erlang[8]. This algorithm, whose execution is triggered by a suited message sent by the Strategy agent, exploits artificial vision techniques and performs a series of transformation (i.e. filtering, threshold, binarisation) on RGB planes of each frame acquired in order to isolate and recognise the required objects. Figure 4 reports some screen-shots of the functioning of the Object Detector. In particular, Figures 4a and 4c show two acquired frames, while Figures 4b and 4d illustrate the filtered images with the objects (respectively a white ball and two blue holes) detected by the agent.

The *Motion Control* agent, which is the only PeriodicFSM type, has the task of controlling the robot's path: it receives, from the Strategy agent, messages containing commands for robot positioning, such as *go to X*,*Y* or *rotate T*, computes the speed of the wheels needed to reach the target, and sends such speeds to the Motion Driver agents. Moreover, in order to ensure that the target is reached, the Motion Control agent periodically requests to Motion Driver the tick count of optical encoders and calculates the absolute position and orientation of the robot [7]. These values are thus compared with the target, making subsequent speed adjustment, if necessary[9]. Another task of the Motion Control agent is obstacle detection. Since the robot has no sensors to detect if an obstacle (e.g. the opponent's robot, a totem, etc.) is in front of it, the Motion Control agent checks if there is no wheel movement within a certain time window (given that wheel's speeds are greater than zero); if this is the case, an obstacle exiting algorithm is started, which entails to move the robot backwards and then rotate it.

The last agent, *Strategy*, is the "brain" of the robot. Being a BasicFSM agent, it is responsible of collecting and putting

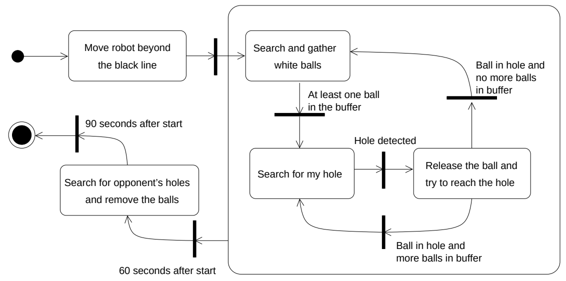

Move robot beyond the black line
Search and gather white balls
Ball in hole and no more balls in buffer
At least one ball in the buffer
90 seconds after start
Hole detected
Search for my hole
Release the ball and try to reach the hole
Search for opponent's holes and remove the balls
Ball in hole and more balls in buffer
60 seconds after start

Fig. 5. Strategy Agent Behaviour

together information about environment and robot subsystems to obtain a valuable and effective playing strategy. Even if the field is mostly immutable (except for the position of totems, which are set before each match) and many of the balls involved are still in fixed position, we chose to implements an intelligent and adaptive strategy instead of a simple "fixed–path" one. For this reason the Strategy agent has to adaptively choose the right action to perform at each time, elaborating data coming from other agents. As Figure 5 illustrates, the very first step of the implemented strategy is "move beyond the first black line", since this guarantees the collection of at least one point[10]. This is performed by suitable commands sent to Motion Control agent. When the black line has been passed, the main strategy loop begins. First the robot looks for white balls and suck them into the buffer: if any white ball is seen by the Object Detector, then the Motion Control agent is issued the commands needed to reach the ball; on the other hand, if no ball has been detected, the Strategy agent tries to search elsewhere, by rotating of a random angle in order to look at other zones of the field. When a ball has been sucked and the Ball Control agent reports the presence of at least one white ball into the buffer, the Strategy agent starts to search a right hole to drop it into (i.e. a hole of the colour assigned to the team, either red or blue), looking at messages from Detector and moving toward a hole as soon as it has been found. When the selected hole is no more visible (i.e. outside the camera scope) a ball is released and, by means of messages coming from Hole Detector, a sequence of commands for fine positioning are sent to Motion Control. If the hole is centred

[8]It uses the OpenCV library [2], which provides a set of fast and optimised image manipulation functions. Proper Erlang-to-C library functions allows this agent to interact with Erlang processes.

[9]Also in this case, a proportional-integrative-derivative software controller is employed.

[10]If the robot does not pass the fi rst black line, then it obtains no points at the end of the match.

and the ball goes into it, the Ball Control agent sends a "Ball Successfully Dropped" message, so the Strategy agent decides to search another hole, if more white balls are present into the buffer, or to look for more white balls. If the ball is not dropped into a given amount of time (for example because of errors in fine positioning) the Strategy agent searches another hole and tries to drop the ball into it. Finally, in the last 30 seconds of game, the Strategy agent tries to find opponent's holes to suck white balls out of them.

V. Implementation Issues

As it has been previously said in the paper, with the exception of the Object Detector, the system has been implemented using the Erlang language. However, even if our research group has realized a FIPA-compliant Erlang agent platform (called eXAT [10], [12], [11], [13], [15], [14]), we did not use it in order to avoid overhead introduced by platform's components for inference, behaviour handling, standard FIPA messaging, etc. This is required in order to have a fast and effective support for agents, rather than the possibility of interacting with other external agents (according to Eurobot rules, the robot must be autonomous and not connected to any network). To this aim, each agent of the robot has been encapsulated in an Erlang process and a suitable library has been developed to support the BasicFSM and PeriodicFSM deadline-aware abstractions. Message passing has been realized by means of the native Erlang constructs to perform inter-process communication (which are designed to be very fast): this resulted in an optimised code able to meet to real-time requirements of the target application.

VI. Conclusions

This paper described the architecture of an autonomous mobile robot, developed by the DIIT Team of the University of Catania to participate to the Eurobot competition. A multi-agent system has been employed for this purpose, composed of several agents in charge of both interacting with physical sensors and actuators, and supporting the game strategy for the robot. A layered architecture has been designed to clearly separate the aspects above—physical world interface and intelligence—and to favour design, modularity and reuse. Due to real time constraints, the system has been implemented using the Erlang language by means of a proper library to support the abstraction needed for using agents in a robotic environment. This allowed us to develop a fast code able to effectively support robot's activities.

VII. Acknowledgements

The authors wish to thank so much all the other components of the Eurobot DIIT Team, who gave a terrific and fundamental contribution in the realization of the robot described in this paper and made this experience not only very useful but also very funny.

These people are **Roberto Di Salvo, Andrea Nicotra, Luca Nicotra, Massimiliano Nicotra, Stefano Palmeri, Francesco Pellegrino, Matteo Pietro Russo, Carmelo Sciuto, Danilo Treffiletti** and **Carmelo Zampaglione**.

Moreover, the authors wish to thank also the official sponsors of the Eurobot DIIT Team, which are **Siatel Srl** (from Catania, Italy) and **Erlang Training & Consulting Ltd**[11] (from London, UK), that, with their support, contributed to make our dream real.

References

[1] "http://www.erlang.org. Erlang Language Home Page," 2004.
[2] "http://opencvlibrary.sourceforge.net/," 2006.
[3] J. Armstrong, B. Dacker, R. Virding, and M. Williams, "Implementing a Functional Language for Highly Parallel Real Time Applications," 1992.
[4] J. L. Armstrong, "The development of Erlang," in *Proceedings of the ACM SIGPLAN International Conference on Functional Programming*, A. Press, Ed., 1997, pp. 196–203.
[5] J. L. Armstrong, M. C. Williams, C. Wikstrom, and S. C. Virding, *Concurrent Programming in Erlang, 2nd Edition*. Prentice-Hall, 1995.
[6] Bollella, Gosling, Brosgol, Dibble, Furr, Hardin, and Turnbull, *The Real-Time Specification for Java*. Addison-Wesley, 2000.
[7] J. Borenstein, H. R. Everett, and L. Feng, *Where am I? — Systems and Methods for Mobile Robot Positioning*. WWW, University of Michigan, USA, http://www-personal.engin.umich.edu/~johannb/position.htm, 1996.
[8] A. Corsaro and C. Santoro, "Design Patterns for RTSJ Application Development," in *Proceedings of* 2^{nd} *JTRES 2004 Workshop, OTM'04 Federated Conferences*. LNCS 3292, Springer, Oct. 25-29 2004, pp. 394–405.
[9] A. Di Stefano, F. Gangemi, and C. Santoro, "ERESYE: Artificial Intelligence in Erlang Programs," in *Erlang Workshop at 2005 Intl. ACM Conference on Functional Programming (ICFP 2005)*, Tallinn, Estonia, 25 Sept. 2005.
[10] A. Di Stefano and C. Santoro, "eXAT: an Experimental Tool for Programming Multi-Agent Systems in Erlang," in *AI*IA/TABOO Joint Workshop on Objects and Agents (WOA 2003)*, Villasimius, CA, Italy, 10–11 Sept. 2003.
[11] ——, "eXAT: A Platform to Develop Erlang Agents," in *Agent Exhibition Workshop at Net.ObjectDays 2004*, Erfurt, Germany, 27–30 Sept. 2004.
[12] ——, "Designing Collaborative Agents with eXAT," in *ACEC 2004 Workshop at WETICE 2004*, Modena, Italy, 14–16 June 2004.
[13] ——, "On the use of Erlang as a Promising Language to Develop Agent Systems," in *AI*IA/TABOO Joint Workshop on Objects and Agents (WOA 2004)*, Torino, Italy, 29–30 Nov. 2004.
[14] ——, "Supporting Agent Development in Erlang through the eXAT Platform," in *Software Agent-Based Applications, Platforms and Development Kits*. Whitestein Technologies, 2005.
[15] ——, "Using the Erlang Language for Multi-Agent Systems Implementation," in *2005 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT'05)*, Compiégne, France, 19–22 Sept. 2005.
[16] P. Dibble, *Real-Time Java Platform Programming*. Prentice Hall PTR, 2002.
[17] I. Infantino, M. Cossentino, and A. Chella, "An agent based multilevel architecture for robotics vision systems." in *Proceedings of the International Conference on Artificial Intelligence, IC-AI '02, June 24 - 27, 2002, Las Vegas, Nevada, USA, Volume 1*, 2002, pp. 386–390.
[18] E. Johansson, M. Pettersson, and K. Sagonas, "A High Performance Erlang System," in 2^{nd} *International Conference on Principles and Practice of Declarative Programming (PPDP 2000)*, Sept. 20–22 2000.
[19] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *JACM*, vol. 20, no. 1, pp. 46–61, Jan. 1973.
[20] Liu, J. W. S., *Real-Time Systems*. Prentice Hall, 2000.
[21] C. Varela, C. Abalde, L. Castro, and J. Gulias, "On Modelling Agent Systems with Erlang," in 3^{rd} *ACM SIGPLAN Erlang Workshop*, Snowbird, Utah, USA, 22 Sept. 2004.
[22] A. Wellings, *Concurrent and Real-Time Programming in Java*. Wiley, 2004.

[11]`http://www.erlang-consulting.com`

Building a MultiAgent System from a User Workflow Specification

Ezio Bartocci*, Flavio Corradini*, Emanuela Merelli*
* Dipartimento di Matematica e Informatica,
Università di Camerino,
Via Madonna delle Carceri, 62032 Camerino, Italy
{*name.surname*}@unicam.it

***Abstract*— This paper provides a methodology to build a MultiAgent System (MAS) described in terms of interactive components from a domain-specific User Workflow Specification (UWS). We use a Petri nets-based notation to describe workflow specifications. This, besides using a familiar and well-studied notation, guarantees an high-level of description and independence with more concrete vendor-specific process definition languages. In order to bridge the gap between workflow specifications and MASs, we exploit other intermediate Petri nets-based notations. Transformation rules are given to translate a notation to another. The generated agent-based application implements the original workflow specification. Run-time support is provided by a middleware suitable for the execution of the generated code.**

I. Introduction

Nowadays open distributed systems, characterized by independent components that cooperate to achieve individual and shared goals, are becoming essential in several contexts, from large scientific collaborations to enterprise information systems. The Grid and agent communities are both developing concepts and mechanisms for open distributed systems, but with different perspectives [10]. Grid community has focused on the main prominent cyberinfrastructures [12] for large-scale resource sharing and distributed system integration, providing tools for secure and reliable resource sharing within dynamic and geographically distributed virtual organizations. Grid computing [9] promises users the ability to harness the power of large numbers of heterogeneous, distributed resources such as computing resources, data sources, instruments and application services. Agent community instead is working on the development of methodologies and algorithms for autonomous problem solvers that can act flexibly in uncertain and dynamic environments in order to achieve their goals [13]. As referred in [10], Grid and Agents need each other and are respectively considered the "brawn" and the "brain" of open distributed systems. With the advent of Grid and Agent-based technologies, scientists and engineers are building more and more complex applications to manage and process large data sets, and execute scientific experiments on distributed grid resources. These applications are generally characterized by the execution of a set of distinct, sometimes repetitive, domain-specific activities. Automating such processes requires a model that describes the coordination of the activities to be executed, the roles involved in the organization and the needed resources. For this reason, during the last decade, the workflow technology has became very important. In fact the Workflow Management Coalition (WfMC) defines a workflow *"the automation of a business process, in whole or part, during which documents, information or tasks are passed from one partecipant to another for action, according to a set of procedural rules."* [21]. In order to provide a proper workflow specification language each standard organization -i.e WfMC, BPMI and OMG- has defined its own process definition language. Our aim is to provide a methodology to translate a User Workflow Specification (UWS) into a MultiAgent System described in terms of Interactive Components [8] (ICs). We have based our approach to describe a MAS with the help of components on that proposed by Ferber in [9] for modelling of MAS in BRIC. Figure 1 shows the two steps of the proposed methodology. In the first step, UWS is translated to a Role-based Workflow Specification (RWS). The user, whose primary expertise is in the application domain, can focus on coordinating domain activities rather than being concerned with the resources involved in the distributed environment. The first translation assigns the resources or roles needed to execute each task. We have chosen Wf-net as high-level specification language suitable to represent the main workflow patterns provided by the most used workflow specification languages -i.e XPDL [22], and BPEL [2]. Wf-net is a well-known extension of classical Petri net [16] notation and it has been introduced in [18]. The second step of the proposed methodology translates RWS into Interactive Components. To describe behavioral aspect of each component we have used BRICs [8] notation; another extension of classical Petri net. We have defined transformation rules to map Wf-net specification patterns into BRICs. This paper is organized as follows.

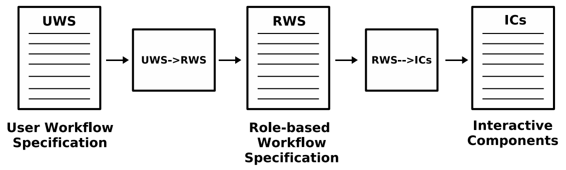

UWS
UWS->RWS
RWS
RWS-->ICs
ICs
User Workflow Specification
Role-based Workflow Specification
Interactive Components

Fig. 1. A two steps methodology

Section 2 describes the background of the work. Section 3 and 4 explain the two steps of our methodology. In Section 5, we present a case study that applies this methodology in Hermes [7] middleware. We conclude in Section 6.

II. BACKGROUND

This section provides some background on Workflow Management System (WMS), Petri nets, High level Petri nets and their application to Workflow Management [18].

A. Workflow Management System

Workflow Management Systems (WMSs) provide an automated framework for managing intra- and interprise business processes. A WMS is defined by WfMC as:*"A system that defines, creates and manages the execution of workflows through the use of software, running on one or more workflow engines, which is able to interpret the process definition, interact with workflow partecipants and, where required, invoke the use of IT tools and applications."* [21]. The most part of implemented WMS are based on a client/server architectural style. In these systems, the workflow enactment is entrusted to a central component, that acts as a server and is responsible for the correct execution. These systems lack the flexibility, scalability and fault tolerance required for a distributed cross-organizational workflow; in fact a monolithic architecture does not allow the execution of workflow or parts of it over distributed and heterogeneous systems. To overcome these limitations, agent-based technology promises to alleviate many of these problems [20] and hence enable adaptive workflow. Moreover, using agent mobility, instances of a workflow or parts of it can migrate; i.e., it is possible to transfer the code and the whole execution state, including all data gathered during the execution, between sites participating in workflow's execution. Agent mobility provides two main benefits. First, migrating workflow decreases efficiently traffic network; usually code implementing workflow specification is less heavy to transfer than the amount of data needed during its execution. The second asset concerns the possibility for the workflow to be executed even in mobile and weekly network connected devices. This model requires a suitable middleware to guarantee code mobility support.

B. Petri nets

A Petri net [16] is a directed bipartite graph with two node types called places and transitions. The nodes are connected via directed arcs. Connections between two nodes of the same type are not allowed. Places are represented by circles and transitions by boxes or bars. According to [15], an *ordinary* Petri net can be defined as a 4-tuple, $PN = (P, T, F, M_0)$ where:

1) $P = \{p_1, p_2, \cdots, p_m\}$ is a finite set of places,
2) $T = \{t_1, t_2, \cdots, t_n\}$ is a finite set of transitions,
3) $F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs (flow relation),

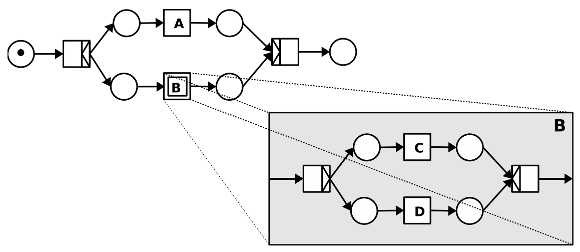


Fig. 2. A subprocess

4) $M_0 : P \rightarrow \mathbb{N}$ is the initial marking function,
5) $P \cap T = \oslash$ and $P \cup T \neq \oslash$.

Ordinary means that all arcs have weight 1.

A place p is called an *input place* of a transition t if and only if there exists a directed arc from t to p. Place p is called an *output place* of transition t if and only if there exists a directed arc from p to t. We use $\bullet t$, $t\bullet$ to denote respectively the set of input places and the set of output places a transition t. The notation $\bullet p$ and $p\bullet$ identifies instead the set of transitions sharing p as input place and as output place respectively.

At any time a place contains zero or more tokens, drawn as black dots. A marking function $M \in P \rightarrow \mathbb{N}$ is the distribution of tokens over places and represents the state of PN. In this definition we do not consider any *capacity restrictions* for places. The number of tokens may change during the execution of the net.

Transitions are the active components in a Petri net: they change the state of the net according to the following *firing rule*:

1) A transition t is said to be *enabled* if and only if each input place p is marked with at least one token.
2) An enabled transition may *fire*. If transition t fires, then t *consumes* one token from each input place p of t and produces one token in each output place p of t.

C. High level Petri nets

A High-level Petri Net (HLPN) [1] is a PN with three main extensions:

- *Extension with color* - in Coloured Petri Net (CPN) [14] tokens are typed and each token has a value often referred as *color*. Transitions determine the values of the produced tokens on the basis of the values of the consumed tokens. Moreover preconditions can be specified taking into account the color of tokens.
- *Extension with time* - using time extension, tokens receive a *timestamp* value that indicates the time from which the token is available. A token with timestamp 10 is available for the consumption by a transition only from moment 10. A transition is enabled only at the moment when each of the tokens

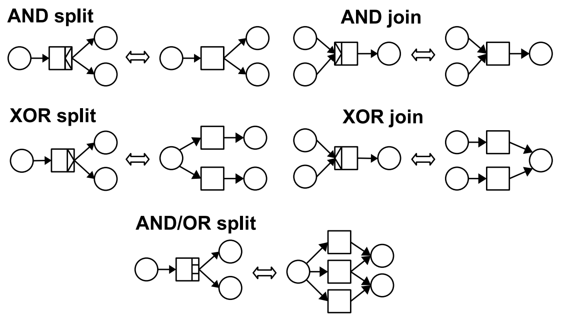


Fig. 3. Special transitions and their translation

to be consumed has a timestamp equal or subsequent to the current time.

- *Extension with hierarchy* - hierarchical extension allows to model complex processes more easily by dividing the main process into ever-smaller subprocesses to overcome the complexity. In this paper, we use the notation proposed by Wil van der Aalst [23], where a subprocess is a transition represented by double-border square as Figure 2 shows.

D. Workflow Nets

Workflow Nets (WF-nets) [23] are a subclass of HLPN where tasks are represented by transitions and conditions by places. A WF-net satisfies two requirements. First of all, it must contain at least two special places: i and o. Place i is a source place with $\bullet i = \oslash$. Place o is a sink place with $o\bullet = \oslash$. Secondly, it must hold that if we add a transition t^* which connect place o with i - i.e. $\bullet t^* = \{o\}$ and $t^*\bullet = \{i\}$ - then the resulting Petri net is strongly connected -from each node there exists a directed path to every other node-. This requirement avoids dangling tasks and/or conditions. In order to make the WF-net suitable for workflow process modelling a set of notational extensions was applied to the standard Petri net definition. In particular, as referred [23], the author of WF-net added to the classical Petri net transition a set of special transitions (AND split, AND join, XOR split, XOR join, AND/OR split), shown in Figure 3 with their translations, to express branching decisions in a more compact and user friendly way.
In the workflow theory [19], routing primitives are defined as a set possible basic patterns that determine which tasks need to be performed and in which order. Using the prevoius defined special transitions as control flow, a set of four basic routing primitives can be obtained as Figure 4 shows:

1) *Sequential routing* - task A is executed before task B,
2) *Alternative routing* - either task A or task B are executed non deterministically,
3) *Concurrent routing* - task A and task B are executed concurrently,
4) *Iterative routing* - task B is repeated

In order to model dependencies between the workflow process and its operative environment three different constructs named "triggers" were added to the standard Petri net -resource, message and time trigger. In this paper we will consider only the resource trigger. In this particular case, a trigger is associated to a specific resource needed to execute a task. As Figure 5 we can consider a trigger as special place linked with the transition representing a task. When the needed resource is not available this place is empty and the transition is not enabled, while if it contains a token it means that the resource is available and the task related to the linked transition could be executed. In the following sections we will consider interactive components as computational resources able to execute tasks under particular cases. The resource trigger can be assigned to every transition and is represented by a small, self-explaining icon (⇓) near the associated transition symbol as Figure 5 shows.

III. Adding roles to Workflow Specification

A workflow process specification defines which tasks need to be executed and in what order. A set of cases, identified by pre- and postcondition, are handled by executing tasks in a specific order. A task which needs to be executed for a specific case is called *work item* [18]. A workflow specification is the composition of both primitive and complex work items. A primitive work item can be directly executed. A complex work item -called subprocess in [18]- must be specified before it can be used; the specification of a subprocess is a workflow of complex and primitive work items. By using subprocesses the specification of workflows is simplified because they enhance both hierarchical specification and reuse: we can use an already existing subprocess without having care of its specification. Work items are generally executed by a *resource* that can be either a machine -i.e. printer or a fax-, a computational entity -i.e. an agent- or a person. Resources are allowed to deal with specific work items. Grouping resources into classes facilitates the allocation of work items to resources. A resource class based on the capabilities of its members is called *role*. A work item which is being executed by a specific resource is called an *activity*. A workflow designer, whose primary expertise is generally in the application domain, should be free to focus on coordinating domain specific activities rather than being concerned with the complexity of a domain specific activity or resources involved to execute it. Users in fact may ignore the topological organization of the distributed environment and resource classes available. The first step of the proposed methodology translates a user workflow specification to a role-based workflow specification. During this step each work item is assigned to a role able to perform it. This operation could be made manually or automatically. In the first case an expert user can assign role by itself, while in the second case an activity repository store all informations about complex activities and the user knows only there is an automatic mapping from domain specific work items and activities. This resource allocation is applied recursively in all work items of each subprocess. Figure 6 shows an example in

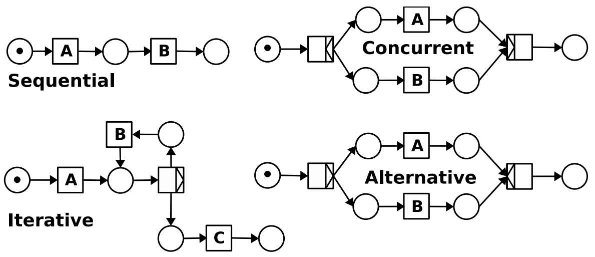


Fig. 4. Routing primitives

bioinformatics. In this case a bioscientist has designed an in-silico experiment -shown on the top of the Figure 6- to globally align some omologous sequences to a given one. This workflow involves five main *work items*:

1) *get_gene_seq* - given a gene id, retrieve the gene DNA sequence,
2) *search_genbank_omologous* - given a DNA sequence, retrieve a set of DNA sequence omologous from NCBI Genbank [3],
3) *search_PDB_omologous* - given a DNA sequence, retrieve a set of DNA sequence omologous from the Protein Data Bank (PDB)[4],
4) *merge_seqs* - merge two or more set of sequences in a set of sequences,
5) *global_alignment* - given a set of DNA sequences calculate the global alignment

In the Role-based Workflow Specification -shown on the bottom of Figure 6- subprocesses *search_genbank_omologous*, *search_PDB_omologous* and *merge_seqs* are substituted with the corresponding set of primitive work items. Each primitive work item is assigned to a specific role. In this case we have three roles A, B and C. Roles are translated into Interactive Components in the next step.

IV. Interactive Components Specification

In the second step the Role-based Specification is translated into Interactive Components. In order to specify the behaviour of each component indipendently from the corresponding generated code, we use BRICs [8], another Petri nets-based notation. In this section we provide transformation rules to translate Wf-net to BRICs notation.

A. BRICs notation

Block Representation of Interactive Components (BRICs) [8] is an high-level language for the design of MultiAgent systems based on a modular approach. A

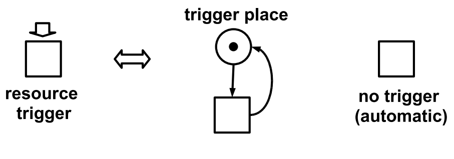


Fig. 5. Resource trigger

BRIC component -see Figure 7 (a)- is a software structure characterized externally by a certain number of input and output terminals and internally by a set of components. Every component is an instance of a class, which describes its internal structure. A structured component is defined by the assembly of the its subcomponents. The input terminals of the structured components are linked to the input terminals of the the sub-components and is also possible to combine terminals of the composite components with sub-components as showed in Figure 7 (b). The behaviour of elementary components is described in terms of a Petri net based formalism. The default net formalism normally used in BRIC is coloured Petri nets with inhibitor arcs. Figure 7 (c) represents the general form of a transition. A transition is defined by *entry arcs*, *exit arcs* and *pre-condition* of activation. Entry arcs are carriers of a condition, in the form of the description of a token including variables. When the place contains a token corresponding to this description, the arc is validated. There are three categories of entry arcs:

1) *Standard arcs*, denoted $a_1, \cdots, a_n$, trigger the transition only if they are all validated consuming tokens which act as triggers and deleting them from input places.
2) *Inhibitor arcs*, denoted $i_1, \cdots, i_m$, inhibit the triggering of the transition if they are enabled without deleting tokens from the input place.
3) *Non-consumer arcs*, denoted $b_1, \cdots, b_k$, work as standard arcs, but they don't delete the input tokens.

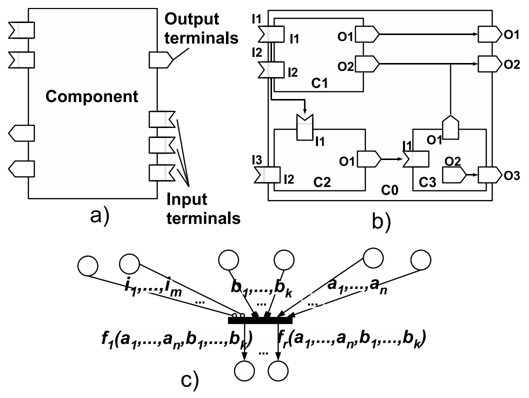


Fig. 7. BRICs notation

An exit arc associate a transition with an output place producing in this position new tokens that depend on the tokens used for triggering the transition. The precondition associated with a transition relates to the external conditions. The components communicate by exchanging information along communication links which connect output terminals to the input terminals. Information is transported through the net in the form of tokens. A token is either an elementary piece of information whose value is a mere presence or absence, or a predicate in the form $p(l_1, \cdots, l_n)$, where each l_i represents a number or a symbol in a finite alphabet. Other importart assumptions concerning this notation are:

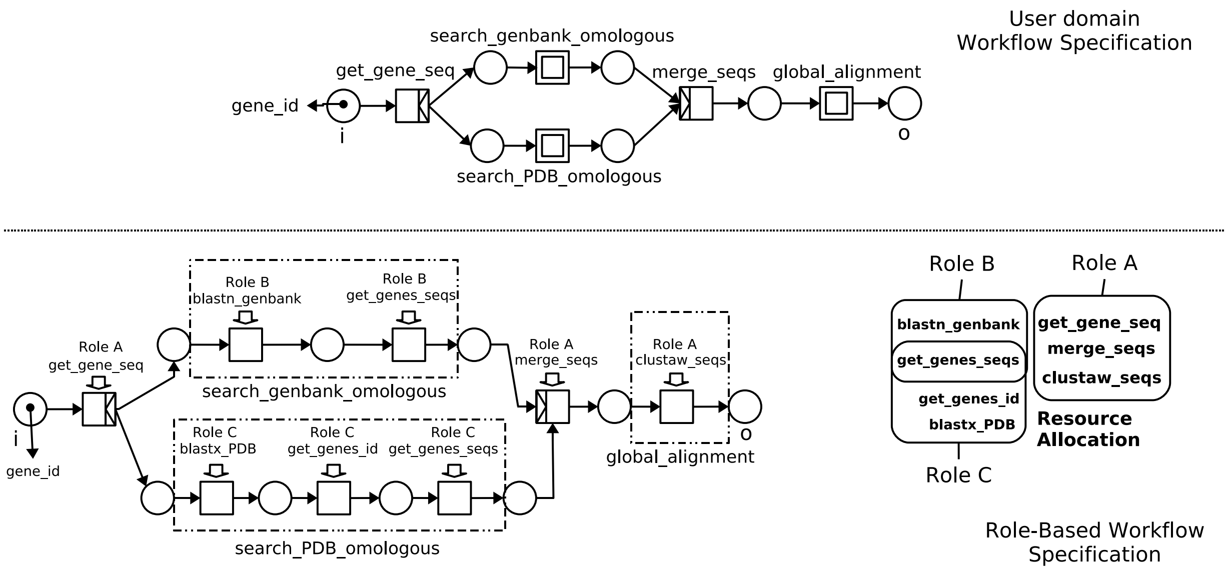


Fig. 6. From User to Role-Based Workflow Specification

1) Input terminals are considered as places, thus names of input terminals are taken to be place identifiers.
2) Any direct link between an input terminal of an incorporating component and an input terminal of incorporated component is assumed to comprise a transition, in accordance with Petri net design rules.

B. Mapping roles with structured components

The translation from a role-based workflow to interactive components specification requires the definition of a structured component skeleton that represents a role-specific implementation. As Figure 8 shows, the basic skeleton has two essential capabilities. First, since it must be able to receive messages from the other external components asynchronously, we specify a subcomponent called *MessagesQueue* that stores messages as coloured tokens following a First In First Out (FIFO) approach. Each message is defined in the form:

```
<sender>: <address> << <Act, Pre, Pa>
```

where *sender* is the identifier of the component sending the message, *address* is the identifier of the component to which the message is addressed. `Act` and `Pre` are respectively the activity to be chosen and the precondition to be set, *Pa* is a possible input parameter for the activity -*null* value means no parameters. In the basic skeleton we specify a second subcomponent, called *Scheduler*, providing, as Figure 9 shows, a set of places and transitions to receive tokens from *MessagesQueue* and to schedule the execution of a set of tasks following the order and cases defined by the role-based workflow specification. Scheduler component has four main places:

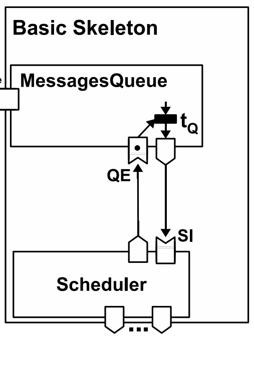


Fig. 8. Basic skeleton component

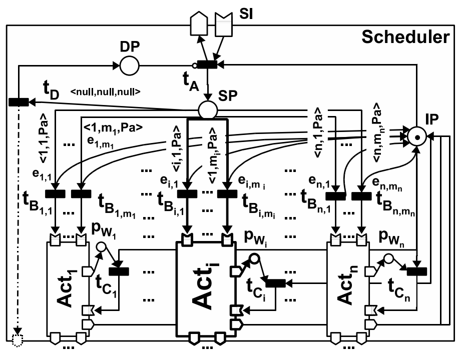


Fig. 9. Scheduler component

1) *Scheduler Input* (SI) - a token in this place means a new message for the scheduler.
2) *Schedule Place* (SP) - after t_A firing produces a coloured token in SP in the form:
   `<Act, Pre, Pa>`
   Each *Scheduler* component contains a set of n *Act* components and $\forall t_{B_{i,k_i}}$ we define an entry arc e_{i,k_i} with the description:
   `<i, k, Pa>`
   where $1 < i < n$, $1 < k_i < m_i$ and m_i is number

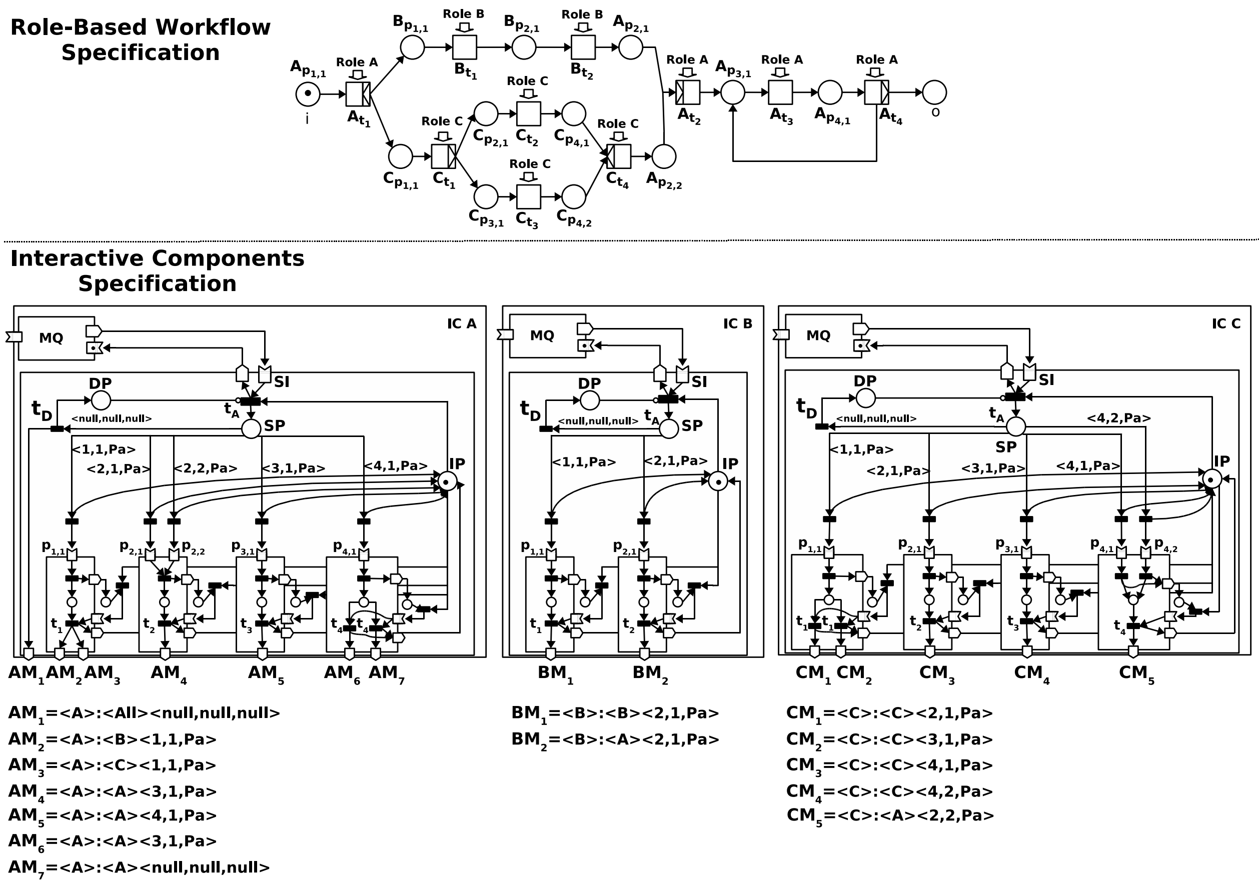


Fig. 11. From Role-Based Workflow to Interactive Components Specification

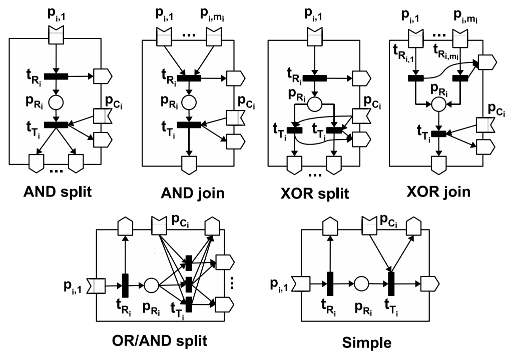


Fig. 10. Mapping activities

of pre-conditions for Act_i. A token in SP matching with a description of an entry arc e_{i,k_i} enables the corresponding transition $t_{B_{i,k_i}}$. The entry arc description for the transition t_D is defined as:

```
<null, null, null>
```

3) *Idle Place* (IP) - when this place contains a token the *Scheduler* is waiting for a new message.
4) *Dead Place* (SP) - the transition t_D when is enabled produce a token in SP inhibiting the transition t_A. Consequently the *Scheduler* can't receive any token in SP place. This place is called dead, because a token here stops the behaviour of this component. When this happens t_D can produce also a token for the external components to stop their behaviour too:

```
<me>: <All> << <null, null, null>
```

C. Mapping activities

An Interactive Component (IC) is an executor of a piece of workflow specification. The final behaviour of an IC is obtained by plugging the activities of the corresponding role into the basic skeleton previously defined. Each primitive activity defined in the Role-based specification is associated with an *Act* component in ICs specification. Figure 10 shows how the routing constructs in Figure 4 are mapped into *Act* components. A component Act_i contains an input terminal for each pre-condition of the mapped activities, which are labelled $p_{i,1}, \cdots, p_{i,m_i}$ where m_i is the number of the activity pre-conditions. When the routing transition t_{R_i} fires the token produced in p_{R_i} enable the task transition t_{T_i} -representing a task to be execute by an IC- is enabled iff IP is not empty. The coloured token produced by t_{T_i} is a message -as previously defined- for its and/or other ICs MessageQueue.

D. An example

Figure 11 shows, on the top a Role-based specification using all possibile routing primitives and on the bottom the translation in ICs specification. For each role in the first corresponds to an IC in the second. All pre-conditions and activities are mapped into Act components adding the right routing transitions and are plugged in the Scheduler of the IC basic skeleton. An Act component produce at least a message describing which are the next IC, Act component and input terminal to be reach and an optional parameter for the task transition. The field *address* in the message specifies which are the receiver IC and an entry arc is assumed from this output teminal and the external input terminal of the IC specified.

V. A CASE STUDY

In the previous sections we have defined a Petri nets-based methodology showing how a user workflow-based application specification can be translated into Interactive Components. As a case study we have applied this methodology in Hermes [7], an agent-based middleware, for the design and the execution of activity-based applications in distributed environment. Hermes is structured as a component-based, agent-oriented system with 3-layer -user, system and run-time- software architecture. Due to the lack of space, middleware architecture is not discussed here and we refer to [7] for further details. In this section, we focus instead on its workflow compiler architecture implementing the methodology previously defined. This component, infact, allows to translate a user domain-specific workflow specification into mobile code supported by Hermes middleware.

A. Workflow compilation process

As Figure 12 shows, workflow compilation process in Hermes requires three main components:

1) *WebWFlow* - allows the user to define graphically a workflow of domain-specific activities. A repository provides a set of complex or primitive activities available for selecting. At this level activity implementation details are hidden to user.
2) *XPDLCompiler* - translates the Role-based workflow specification into Interactive Components specification and generates the code to be executed on Hermes middleware. In this case another repository provides the implementation of each activity as a code template.
3) *Hermes middleware* - supports the generated code execution and mobility.

In the follow sections we focus on the first two components details.

B. WebWFlow

WebWFlow is a web-based workflow editor supporting the workflows specification by composing activities in a graphical environment. The graphical notation provided is mapped by WebWFlow into an XML Process Definition

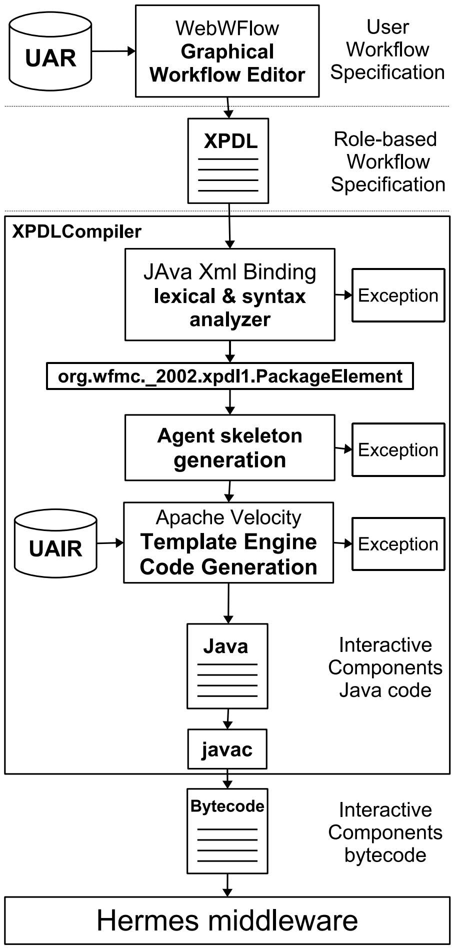


Fig. 12. Workflow compilation process

Language (XPDL) [22] document. WebWFlow allows to import complex activities from the User Activity Repository (UAR). This repository contains the role-based definition of domain-specific activities. The implementation of each activity in UAR is provided instead by the User Implementation Activity Repository (UAIR) and corresponds to a piece of Java code extended with Velocity Template Language(VTL)[11]. The XPDL produced by WebWFlow is a Role-based workflow specification.

C. XPDLCompiler

XPDLCompiler receives an XPDL document and generates the Java bytecode implementing Interactive Components. A lexical and syntax analyzer performs the validation and the parsing of the XPDL document using the Java Architecture XML Binding [17]. After this first phase, the compiler checks if the activities used in the workflow specification have a corresponding implementation in UAIR. Each role is translated in an Agent skeleton, an extension of Hermes UserAgent Java class. As Figure 13 shows, a UserAgent provides the needed communication methods to interact with other UserAgents. Then, for each activity, the corresponding

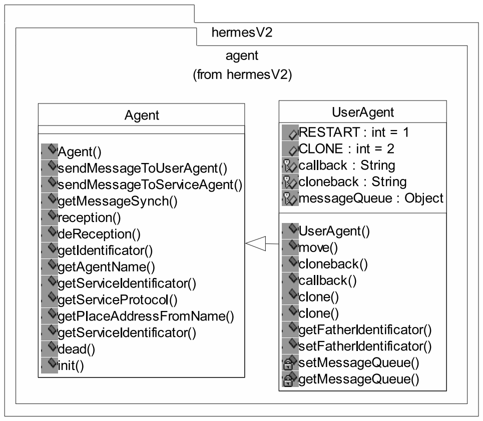


Fig. 13. UserAgent and Agent main methods

implementation code in UAIR is plugged into an Agent skeleton and each internal scheduler is set. The Java code generation is performed using Apache Velocity (http://jakarta.apache.org/velocity/) template engine. Finally, using the Java compiler, the generated bytecode can be loaded into Hermes middleware.

Acknowledgment

This work is supported by the Investment Funds for Basic Research (MIUR-FIRB) project Laboratory of Interdisciplinary Technologies in Bioinformatics (LITBIO). We would also like to thank Luca Tesei for his valuable remarks and suggestions.

VI. Conclusion

This paper presents a methodology to build a MultiAgent System described in terms of Interactive Components from a domain-specific User Workflow Specification. The whole approach is described using Petri nets-based notation. This provides many benefits. Petri nets are well-studied formalisms and there are many tools available for verification. The high-level of description provided by Petri Nets guarantees independence with vendor-specific process definition languages. Behaviour of agents can also be described using BRICs, another Petri nets-based notation. In this case it is possible to describe components independently from the their implementing code. Using transformation rules from a notation to another we reduce the gap between workflow specifications and MultiAgent System. Our approach currently supports the building of a MAS based on message passing communication, its extension towards uncoupled communication will be next considered. As future work we also aim to use the approach proposed in [5], [6] to validate the implementation starting from the model.

References

[1] W. Aalst. Putting Petri nets to work in industry. *Computers in Industry*, 25(1):45–54, 1994.
[2] T. Andrew, F. Curbera, H. Dholakia, Y. Goland, and et al. Business process execution language (bpel) for web services version 1.1. Technical report, IBM, 2003.
[3] D. Benson, I. Karsch-Mizrachi, D. Lipman, J. Ostell, and D. Wheeler. Genbank. *Nucleic Acids Res.*, 34:D16–20, 2006.
[4] H. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. Bhat, W. H., S. I.N., and B. P.E. Wildfire: distributed, grid-enabled workflow construction and execution. *Nucleic Acids Res.*, 28(1):235–42, 2000.
[5] A. Bertolino, F. Corradini, P. Inverardi, and H. Muccini. Deriving test plans from architectural descriptions. In *ICSE*, pages 220–229, 2000.
[6] A. Bertolino, P. Inverardi, and H. Muccini. An explorative journey from architectural tests definition downto code tests execution. In *ICSE*, pages 211–220. IEEE Computer Society, 2001.
[7] F. Corradini and E. Merelli. Hermes: agent-based middleware for mobile computing. In *Mobile Computing*, volume 3465, pages 234–270. LNCS, 2005.
[8] J. Ferber. *Multi-Agent System: An Introduction to Distributed Artificial Intelligence.* Addison-Wesley, 1999.
[9] I. Foster and C. Kesselman. *The Grid: Blueprint for a Future Computing Infrastructure*. Morgan Kaufmann Publishers, San Francisco, CA, 1998.
[10] I. T. Foster, N. R. Jennings, and C. Kesselman. Brain meets brawn: Why grid and agents need each other. In *AAMAS*, pages 8–15. IEEE Computer Society, 2004.
[11] A. Group. Vtl reference guide. http://jakarta.apache.org/velocity/docs/vtl-reference-guide.html.
[12] T. Hey and A. E. Trefethen. Cyberinfrastructure for e-Science. *Science*, 308(5723):817–821, 2005.
[13] N. R. Jennings. An agent-based approach for building complex software systems. *Commun. ACM*, 44(4):35–41, 2001.
[14] K. Jensen. *Coloured Petri Nets. Basic concept, analysis methods and practical use*. EATCS monographs on Theoretical Computer Science. Springer-Verlag, Berlin, 1996.
[15] T. Murata. Petri nets: Properties, analysis and applications. In *Proceedings of the IEEE*, volume 77, pages 541–580, April 1989.
[16] C. A. Petri. *Kommunikation mit Automaten.* PhD thesis, Institut für instrumentelle Matematik, Bonn, 1962.
[17] Sun. Java architecture for xml binding (jaxb). http://java.sun.com/webservices/jaxb/.
[18] W. van der Aalst. The application of petri nets to workflow management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.
[19] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.
[20] J. M. Vidal, P. A. Buhler, and C. Stahl. Multiagent systems with workflows. *IEEE Internet Computing*, 8(1):76–82, 2004.
[21] WfMC. Workflow management coalition terminology and glossary. Technical Report WFMC-TC-1011, Workflow Management Coalition, 1999.
[22] WfMC. Xml process definition language (xpdl). WfMC standard, W3C, October 2005.
[23] K. v. H. W.M.P. van der Aalst. *Workflow Management - Models, Methods and Systems*. MIT Press, Cambridge, 2002.

Agent-Based Virtual Communities for Interactive Digital Television

Federico Bergenti, Lorenzo Lazzari, Agostino Poggi
Dipartimento di Ingegneria dell'Informazione
Università degli Studi di Parma
Parco Area delle Scienze 181/A, 43100 Parma, Italy
{bergenti, lazzari, poggi}@ce.unipr.it

Abstract

This paper describes a multi-agent framework designed to support the creation and effective management of virtual communities in an Interactive Digital Television (IDTV) scenario. The possibilities that this framework offers are demonstrated by means of two sample applications: a real-time community game and an asynchronous auction. For the sake of completeness, the paper also presents an overview of IDTV technologies.

1. Introduction

Nowadays, a migration from analogue to digital TV is taking place in TV. This change has two main implications: the capability to broadcast more channels in the same bandwidth, and the possibility to send software applications mixed with audiovisual contents. These two great advantages have permitted the great diffusion of this new technology, which is becoming a new power means to develop new types of services.

In this paper we present our multi-agent framework, developed starting from the idea to integrate the technology of Interactive Digital Television (IDTV) with the concept of virtual community, which we can define as a technology-supported cyberspace, centered upon communication and interaction of participants, resulting in a relationship being built up. So, with this type of integration, our aim is to offer to IDTV users, a range of services (such as multiplayer games, on-line auctions, etc.) which are very common if we think to the idea of virtual community related to the Web.

In this way, the potentialities of interactive DVT can enormously grow allowing its users to take advantage of a new number of useful applications and moving the concept of interactivity from the simple interaction user-application to a new type based on the cooperation among a wide number of users.

2. IDTV

Interactive TV is a technology which combines broadcast video, broadcast radio, computing power and the Internet. This combination of different mediums and services provides the viewer with a new experience. This is possible because of an ongoing transition from analogue TV to digital TV.

We can clearly say that the digital technology is driving television towards a new world of amazing possibilities, where spectator is no longer limited to observe contents selected by the operator. More and more, new dynamic and interactive services are being introduced in everyday digital TV: complementary information to audio-visual contents, electronic program guides, selection of properties in configurable contents (language, camera angle or particularized advertisement), pay-per-view, etc. So we can consider the term "interactivity" as the possibility for the consumer to actively influence the behavior of broadcasted television, services and applications. This can be accomplished, for example, by means of a remote control for channel hopping, by fetching information via teletext or by sending data via an interaction channel. This all creates a context, which allows to have a mutual influence between the viewer, broadcaster and application provider.

The interactive TV technology, as we will see in next section, is based on the broadcasting of a digital transport stream which permits operators to mix traditional audio-visual contents with binary data, so making possible to deliver multimedia applications to be executed in a digital TV or in a set-top box. These applications, synchronized with audio-visual contents, adapt themselves to spectator characteristics, implement interaction with users and provide return channels for communication with content providers.

The Multimedia Home Platform [1] is a standard published by the DVB (Digital Video Broadcasting) consortium in 2001, which consists of a combination of broadcast and Internet, offering a common Application Programming Interface (API) accessible for everyone who wants to develop applications, set-top boxes, television devices or the combination of all.

Fundamentally the MHP standard defines a generic interface between interactive digital applications and the terminals on which those applications execute. This interface decouples different provider's applications from the specific hardware and software details of different MHP terminal implementations.

The MHP extends the existing, successful DVB open standards for broadcast and interactive services in all transmission networks including satellite, cable, terrestrial, and microwave systems.

The applications downloaded to the MHP terminals, typically set-top boxes, are Java applications called Xlet, built on a suite of APIs tailored specifically for the interactive TV environment: Java TV APIs [2], HAVi (user interface) [5], DAVIC APIs [4] and DVB APIs [3].

The 1.1 version of the standard defines three profiles:

1. Enhanced Broadcast: it is the basic profile which only allows the enrichment of the audio-video contents with information and images which can be viewed and navigated by users on the TV screen:
2. Interactive Broadcast: it is the intermediate profile that uses the set-top box return channel to supply services with a higher level of interactivity. In fact this profile supports the loading of MHP applications not only through the broadcast channel but also through the return channel;
3. Internet Access: this profile, using the return channel, allows the user to access to the Internet contents.

As we can understand from the previous description of the MHP levels, the interactive TV paradigm is based on two different channels: a broadcast channel from the application/contents provider to the set-top box and a return channel (dial-up, GPRS, ADSL, Ethernet, etc.) from the set-top box to the provider.

Figure 1 shows the use of a carousel to continue play-out a Java application. The application and the corresponding audio-visual material are then multiplexed to form a single MPEG-2 transport stream.

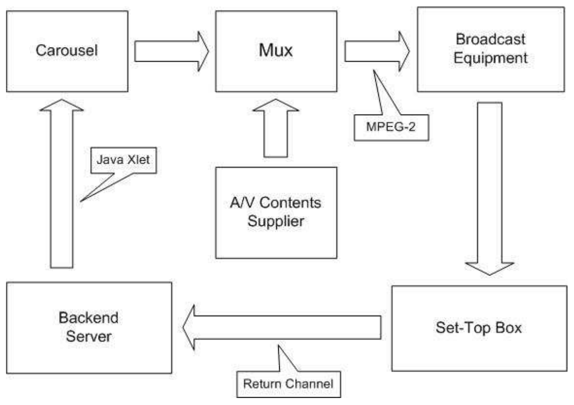


Figure 1. The interactive broadcasting chain

The resulting broadcast is received and decoded by the set-top box, the audio-visual content played and the Java application run.

Subsequent user interactions with the application lead to information being sent via the return channel to a back-end server. Depending on the application, this information may result in modifications to the current application content (i.e. voting information) or stored for later processing in a database present on the server (i.e. for an online shopping application).

About the transport, in digital TV MPEG-2 is not only a standard for encoding audio and video, but it is also used as the means by which raw data and applications are transported in the broadcast stream. In particular, DVB has extended the traditional scheme and way to use MPEG-2 for MHP by specifying how to embed a Java application within the stream, this includes information on how to specify the main class, class search path and the application argument list etc.

Although MPEG-2 provides a means of transporting the Java applications along the audio-visual content, to support the possibility that the user may change channel and select the Java program at any point of the transmission, the same application has to be broadcasted in loop. This is exactly what a broadcast carousel does: it keeps playing the same application around and around. The application is continuously multiplexed with the audio-visual content for the transmission, to allow the viewer to access to the interactive TV application whenever he wants.

About the applications, as we said previously, we have Java applications, but they are not complete Java applications in the normal sense. These applications are much more like applets in that they are loaded and run by a life cycle manager residing on the set-top box.

3. MHP-based Virtual Communities

In spite of the great research interest collected in the last years and the high number of functionalities already supported, in these days the research groups that work on the IDTV MHP standard are focusing their interest especially on the personalization of the IDTV contents on the base of the analysis of the user profile and preferences.

In accordance with our point of view, at the moment what is totally absent it is the collaborative aspect, that is the integration, in the digital television technology, of particular types of services to support groups of users joined by particular types of interests or necessities. These types of services are very common on the Web, we can think about the enormous number of forums, of blogs or of general services which allow a direct interaction among their users (on-line auctions, multiplayer games, etc.).

So the starting point from which our project has risen has been the aim to enrich the IDTV paradigm based on MHP and described in the previous sections with the introduction of the concept of "virtual community" very common on the Internet network.

A generally agreed upon definition of a virtual community would be a good starting point. What we need is a working definition of the virtual community, a consensus found in the major stream of literature, a definition that understood by most of people.

In his definition of a virtual community, Howard [6], the primary early advocator of virtual communities and often quoted in the literature, includes factors that describe a virtual community as a social aggregations that emerge from the Net when enough people carry on those public discussions long enough, with sufficient human feeling, to form webs of personal relationships in cyber-space. Hagel and Armstrong [7] focus on the content and communication aspects with special emphasis on member generated content: for them virtual communities are computer-mediated spaces where there is a potential for an integration of content and communication with an emphasis on member-generated content. The definition from Jones and Rafaeli [8] uses the term "virtual public" instead of virtual community. In particular, they say that virtual publics are symbolically delineated computer mediated spaces, whose existence is relatively transparent and open, that allow groups of individuals to attend and contribute to a similar set of computer-mediated interpersonal interactions. Another interesting point of view is the Romm and Clarke's [9] definition, which points out only the aspect of communication, that is via electronic media: virtual communities are groups of people who communicate with each other via electronic media, rather than face to face.

In literature we can find a lot of other definitions, but we can find some common aspects. The first similar point is cyberspace. All of the definitions state that the virtual community should be on the net, use computer-mediated spaces, or cyberspace. This point differentiates the virtual community from a real community. The second aspect in common is the usage of technology to support the activities in the virtual community. The different definitions directly or indirectly emphasize that access to the virtual community is through the computer or electronic media, i.e., technology. The third similar aspect is that the content or topics of the virtual community are driven by the participants. As mentioned, the participant driven community, not the web site coordinators, clearly distinguishes the virtual community from online information services. The final shared aspect is the successful virtual community relationship culminating after a certain period of communicating together.

To sum up, a working definition of a virtual community could be: a technology-supported cyberspace, centered upon communication and interaction of participants, resulting in a relationship being built up.

With our framework, in which the idea of virtual community is integrated with the interactive digital TV technology, we focus our interest especially on the second of the common aspects that define the virtual community concept: the support technology. In fact, we increase the horizons and the possibilities of the virtual communities by giving new types of services based on a new and more user-friendly technology like the IDTV.

In fact, the possibility to integrate the increasing IDTV technology with the idea of virtual community can give two great profits: on one hand we have a large increase of the digital television potentialities, opening new ways of communication and new types of services for the IDTV users; on the other hand, consequently, we give the possibility to enter in a virtual community taking advantage of his services also to a user range, the IDTV users, that sometimes can have not enough ability to surf the Web.

We can say that the integration of the digital television with the paradigm of virtual communities can extend the basic concept of interactivity, moving it from a simple logic user-TV to a more interesting logic based on the interaction user-user or user-community of users.

In particular, the ideas at the base of the development of our framework have been principally two: the support for community games and a more wide support for virtual communities involved in cooperative activities such as on-line auctions.

The technology used is a multi-agent technology, this because the intrinsic characteristics of multi-agents systems and of the agents themselves, such as proactivity, make them very proper to our scope.

4. The Framework

Agents need resources to act and to communicate. In FIPA [10] specifications, the run-time support providing such resources is the agent platform. Agents can run only in the scope of an agent platform providing the basic services to support interoperability: a means for sending and receiving messages and a means for finding agents, i.e., white pages and yellow pages. We do not request the platform to provide any support for concepts from agent-oriented software engineering such as autonomy or service-level interoperability. Basically, the platform is only meant to support the typed-message agent model.

Agents communicate explicitly sending messages and such messages may reach either agents within the same platform or agents on different platforms. This difference must be transparent to the developer and a fundamental characteristic of agent platforms is enabling this to support open societies where agents running on different platforms can join and leave dynamically.

The distribution and cooperation of agents residing on different platforms implies the conformance to a standard. At the moment, only FIPA is producing specifications for agent platforms.

At the moment, a number of FIPA platforms are available [11, 12, 13, 14], our middleware is developing the enabling technology for allowing the seamless deployment of agents to the Java-enabled IDTV devices such MHP-compliant set-top boxes.

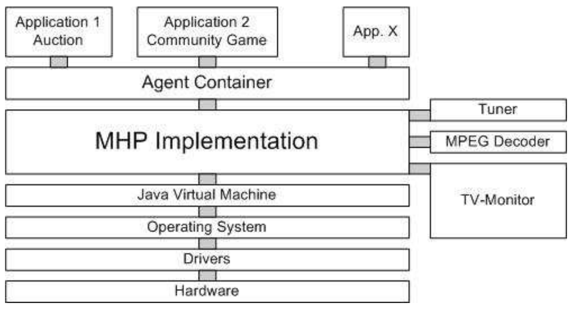


Figure 2. Client side architecture

Our framework is deployed as a multi-agent platform which we can split in two main sides: a server and a client side. The server side is set on a web server and it is deployed using the standard FIPA specifications, instead the client side is the more innovative one, because, since it is set on the set-top box, it requires to enable FIPA Agents on these types of devices.

In the next sections we give a first description of the platform architecture, starting from the client side, and then we will talk about the behaviour of the global platform, giving some example of virtual communities support.

4.1. Client side

The agent container set on the client side must be flexible enough to allow the integration of new services for the virtual community users. For this reason, we think that the best choice is to conceive the client-side of our framework as a MHP interactive application.

In the DVB MHP standard, applications are executed in the context of concrete services or events in a service, and, usually, they do not survive after finishing that context. In order to support services for virtual communities, we have to take into account that our system needs to store all the viewers' preferences about a particular topic (i.e. the user profile in a community game). So our approach integrates a special agent, named User Agent, which has the basic roles to work as an interface between the user and the rest of the system and to store the user preferences.

The User Agent is responsible of building the user profile, maintaining it when its user is on-line and notify to the system when his related user is active. The communication UA-user is performed by a standard GUI by which the user can manage his profile and the different services. Clearly, on the other side, the communication between the UA and other agents is based on FIPA specifications.

In order to support particular services for virtual communities, such as the possibility for a user to delegate to her/his personal agent the negotiation of a price in an on-line auction, this basic type of agent is always active on the user device.

The framework allows the development of other types of agents to guarantee other particular types of services, but for the moment our idea of the client side is that it must be based on "thin" software, so the reasoning mechanisms for the moment are delegated to the server side agent platform.

4.2. Server side

The server side of our framework consists in an agent container set on a standard Web server connected with the clients through the return channel of the set-top boxes.

In order to support services for virtual communities, the server side of the system has to include at least five different types of agents: a SP Agent (Set-top box Proxy Agent), a MP Agent (Mux Proxy Agent), a User Profile Manager, one or more Service Agent and a Directory Facilitator.

The SP Agent represents the interface between the server side multi-agent architecture and the client-side device: this agent receives the requests which came from the User Agent set on the user set-top box and manages them interacting with other kinds of agents.

On the other side, we have another proxy agent, called MP Agent, which is responsible to update the state of the application and to notify it to the Multiplexer, in order to update the raw data related to the Xlet embedded in the MPEG-2 stream and, consequently, the state of the interactive application displayed on the user's TV screen.

Between the two proxy agents have a specific kind of agent, named Service Agent, which is responsible of a particular type of service offered by the framework to the virtual organization. In example, if we think to a multi-player game, the Service Manager related to this type of service will be responsible to manage the state of the game, to find one or more appropriate partners to play, etc.

The User Profile Manager agent is responsible of maintaining the profile of the users and the information/preferences of the users themselves in relation to the particular types of services offered by the system (i.e. game preferences, skill level, etc.).

In the end, the Directory Facilitator is responsible to inform an agent about the address of the other agents of the system.

Figure 3 gives a graphical representation of the architecture of the system, focusing both on the interactions between agents and between the different devices. In figure 3 groups of three agents means that there can be one or more agents of that type.

5. Sample Services

In this last section we give some example of services supported by our system. In particular, as we said when we introduced our framework, the ideas at the base of the development of our system have been principally two: the support for community games and a more wide support for virtual communities involved in cooperative activities such as on-line auctions.

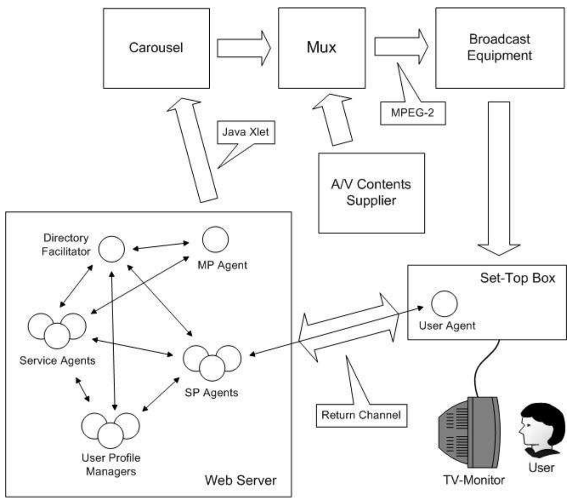


Figure 3. Architecture of the system

5.1. Community games

The idea to play a game in a virtual way with other people connected by a network or, in general, by a technology supporting the real-time interaction between the game participants is very common and diffuse on Internet. With our system we match this idea with the TDV interactive television, allowing IDTV users to play a community game without using any type of computer and of network, but through their IDTV device.

To describe quickly the system behavior relatively to such type of service, we can consider a simple type of game like "Othello", which requires two players. When an IDTV user wants to play an Othello match versus another user he has fundamentally to complete two steps before starting the match: the service configuration and the choice of the opponent. The service configuration is a task that the user has to perform only the first time she/he uses the application: the user has to insert some information like the game preferences, the skill level, etc. Once the game has been configured, the User Agent communicates them to the server side of the system to update the user profile managed by the User Profile Manager agent.

At this point the user is able to play: when she/he run the game by his set-top box, the User Agent notifies the server-side that his associated user wants to play. At this point the Service Agent related to that game creates a new game instance and the User Profile Manager agent find a possible opponent (the other user has to be "on-line" and has to be a compatible skill level).

Once the opponent has been chosen, the match can start: the system, e.g. the Service Agent, continuously updates the state of the application in relation to the moves made, one after the other, by the participants until the end of the match. Obviously, in relation to the result, the system updates users' profiles.

5.2. Online auctions

Also the paradigm of the on-line auctions is very common for the Web users, we can think about the famous eBay Web site to quickly understand the enormous success that these types of services have collected in the last years. The behavior of the system is very similar to the previous case, in the sense that also for this type of service the user has to make an initial configuration of the application inserting her/his data which are used to update his profile.

Differently from the community game, in this type of service we have not a real-time interactions among the involved users but we have an asynchronous communication. When a user wants to sell something she/he opens a new auction inserting the initial price, the deadline, etc., then the User Agent notify the server side of the system and the Service Agent related to this type of service creates a new auction instance. From this moment all the users using this type of service can participate to the auction making their offers or selecting a maximum budget and delegating to their related User Agent the task. Once the auction has expired, the Service Agent deletes the related auction instance and the User Profile Manager agent updates the user profiles related to the involved users.

6. Conclusions

This paper presents a multi-agent framework that we realized to support the effective and fruitful implementation of virtual communities in an Interactive Digital Television scenario. Our framework gives IDTV application developers and service providers the possibility of running virtual communities to support the realization of interactive end-users applications, e.g., real-time multiplayer games and on-line auctions.

We strongly believe that the tight integration between IDTV and virtual communities that our framework provides can put a new perspective on IDTV. On the one hand, our framework opens new ways of communication and new types of services for the IDTV users and, on the other hand, it expands enormously the range of users that are possibly reached by everyday Internet-based virtual communities.

At the moment, our framework is under development. For the server-side of our multi-agent system we are using JADE (Java Agent DEvelopment Framework) [11, 15], which is a software framework to aid the realization of agent applications in compliance with the FIPA specifications for interoperable intelligent multi-agent systems. Client-side is based on new, yet somehow consolidated, IDTV technologies, e.g., MHP.

Our future work is related to the development of new types of applications and services expanding the functionalities and the multi-agent architecture of the framework.

References

[1] MHP Site *http://www.mhp.org*

[2] JavaTV Site *http://java.sun.com/products/javatv*

[3] DVB Site *http://www.dvb.org*

[4] DAVIC Site *http://www.davic.org*

[5] HAVi Site *http://www.havi.org*

[6] R. Howard. *The Virtual Community: Homesteading on the Electronic Frontier*. Addison Wesley, 1993.

[7] J. Hagel, A. Armstrong. *Net Gain: Expanding Markets through Virtual Communities*. Harvard Business School Press, 1997.

[8] Q. Jones, S. Rafaeli. "Time to Split, Virtually: 'Discourse Architecture' and 'Community Building' as means to Creating Vibrant Virtual Metropolises". *Int'l J. Electronic Commerce & Business Media*, 10(4), 2000.

[9] C. Romm, R. J. Clarke. "Virtual Community Research Themes: A Preliminary Draft for A Comprehensive Model". *Procs. 6th Australasian Conference on Information Systems*, 1995.

[10] FIPA Site *http://www.fipa.org*

[11] F. Bellifemine, A. Poggi, G. Rimassa. "Developing Multi-agent Systems with a FIPA-compliant Agent Framework". *Software Practice and Experience*, 31:103-128, 2001.

[12] J. Heecheol, C. Petrie, M. R. Cutkosky. "JATLite: A Java Agent Infrastructure with Message Routing", *IEEE Internet Computing*, Mar./Apr., 2000.

[13] C. Petrie. "Agent-based Engineering, the Web, and Intelligence", *IEEE Expert*, 11(6), 1996.

[14] F. Bergenti, A. Poggi, B. Burg, G. Caire. "Deploying FIPA-compliant Systems on Handheld Devices". *IEEE Internet Computing*, 5(4):20-25, 2001.

[15] JADE Site *http://jade.tilab.com*

Distributed Workflow Enactment: an Agent-based Framework

Giancarlo Fortino, Alfredo Garro, and Wilma Russo

Abstract—**This paper describes the design and the implementation of an Agent-based Workflow Enactment Framework (AWEF) which can be instantiated on the basis of a workflow schema for obtaining a specific workflow enactment engine. A workflow engine therefore is a MAS capable of managing instances of the workflow schema used for the instantiation of AWEF. Each MAS adopts a hierarchical organizational structure composed by an *EnacterAgent*, which is responsible of the activation and monitoring of the workflow, one or more *ManagerAgent*s, which are responsible of the execution and control of the workflow/subworkflows according to a parent/child model, and one or more *TaskAgent*s, which are responsible of the execution of internal tasks and/or of the wrapping of external tasks or services. The hierarchical distribution of the workflow execution control between the *ManagerAgent*s and the distribution of the computation among the *TaskAgent*s allow for more flexible, efficient, and robust enactment services.**

Index Terms—**Multi-Agent Systems, Distributed Workflow Enactment, Workflow Patterns, Agent-based Applications.**

I. Introduction

WORKFLOW Management Systems (WFMS) are systems designed to automate complex activities consisting of many dependent tasks [26]. In the last decades WFMS have been developed to provide support to the modeling, improvement and automation of business management, industrial engineering, and data-intensive scientific processes [25,10]. Since each business area can benefit from workflow management it is possible to distinguish different kind of workflows and related workflow management techniques specifically conceived for meeting the requirements of a specific business area and fully supporting the associated business processes. A main distinction that can be done is between Collaborative and Production-oriented workflows. The former are information centric: human interactions drive the execution of workflows in a loosely structured manner. In this case WFMS are Computer Supported Cooperative Work (CSCW) systems that offer groupware applications and other shared workspace tools for supporting human interactions [4]. Instead, Production workflows are process-driven due to their highly repetitive nature [7]. In this case the processes are highly structured and the adopted WFMS are based on workflow enactment services able to offer an efficient and accurate control about the flow of the processes. Moreover, in a Production workflow, most of the tasks are executed automatically by software programs and applications without interacting with human users. Such a kind of workflows can be modeled as a set of interrelated services (Service Workflow) [29]. In the context of Internet-based workflows [19], such services are distributed and owned by different organizations so that they could become unavailable due to the lack of network service guarantees. To deal with this important issue, a dynamic service allocation of services is often required as well as the negotiation of service level agreements (SLA). Due to these reasons the enactment of Internet-based workflows requires more flexible enactment engines based on more adequate coordination mechanisms.

To effectively fulfill such requirements the Agents paradigm and technology are being used since Agents are widely considered very suitable for the modeling and implementation of complex software systems in open distributed environments [18]. In particular, in the context of workflow management, the use of the Agents paradigm allows for transforming a workflow from a sequence of activities, that are often modeled and consist of (Web) services invocation, in a society of proactive, autonomous and coordinable entities (or multi-agent system) whose coordinated interactions drive the workflow execution [20,17].

This paper proposes an agent-based approach for the distributed enactment of workflows. The workflow enactment is enabled by an Agent-based Workflow Enactment Framework (AWEF) which is instantiated on the basis of the schemas of the workflows to be enacted so obtaining specific workflow engines. A workflow schema can be defined by using the Workflow Patterns identified and proposed by van der Aalst [25] and can be represented using YAWL (Yet Another Workflow Language) [24]. A workflow engine therefore consists of a MAS capable of managing instances of the workflow schema used for the instantiation of the workflow engine. The framework provides the base agents for workflow enactment (*EnacterAgent*, which is responsible of the activation and monitoring of the workflow,

G. Fortino is with the Department of Electronics, Informatics and Systems (DEIS), University of Calabria, Rende (CS), 87036 Italy. (e-mail: giancarlo.fortino@unical.it).

A. Garro is with the Department of Electronics, Informatics and Systems (DEIS), University of Calabria, Rende (CS), 87036 Italy. (e-mail: alfredo.garro @unical.it).

W. Russo is with the Department of Electronics, Informatics and Systems (DEIS), University of Calabria, Rende (CS), 87036 Italy. (e-mail: wilma.russo @unical.it).

ManagerAgent, which is responsible of the execution and control of the workflow, and *TaskAgent*, which is responsible of the execution of internal tasks and/or of the wrapping of external tasks or services), their interaction protocols and a set of control-flow classes which are associated to the behavior of the *ManagerAgent* and implement the Workflow Patterns. The framework is implemented by using JADE [3,16], a FIPA-compliant [13] Java-based agent development environment which basically offers a distributed agent platform and an API for agent programming.

The remainder of the paper is organized as follows. Section 2 introduces some background concepts about workflow enactment, presents our agent-based approach enabled by AWEF and reports some related works. Section 3 and Section 4 describe the design and the JADE-based implementation of AWEF respectively. Finally conclusions are drawn and directions of further work delineated.

II. Distributed Workflow Enactment

The Workflow Reference Model, proposed by the Workflow Management Coalition (WfMC) [28], describes a generic architecture for workflow management consisting of several functional components interfaced with a Workflow Enactment Service (see Figure 1). The Process Definition Tools allow a process designer to define business processes often adopting a diagrammatic representation. The diagrams (or workflow schemas), represented in a Process Description Language (PDL), are received by the Workflow Enactment Service via Interface 1. The Workflow Client Application is usually the application which requests the enactment of a workflow to the Workflow Enactment Service by specifying the workflow schema to be enacted and passing the parameters (Case Activation Record) needed for the activation and execution of a specific workflow instance. During its enactment a workflow can be administered and monitored (Interface 5) and it may interact with other automated business processes (Interface 4), with human participants (Interface 2) and with other applications without human intervention (Interface 3).

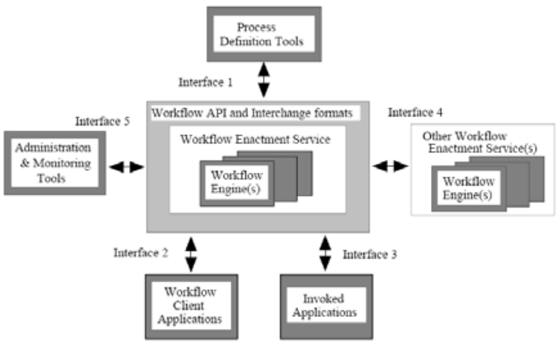

Process Definition Tools
Interface 1
Workflow API and Interchange formats
Interface 4
Interface 5
Workflow Enactment Service
Other Workflow Enactment Service(s)
Administration & Monitoring Tools
Workflow Engine(s)
Workflow Engine(s)
Interface 2
Interface 3
Workflow Client Applications
Invoked Applications

Fig. 1. The reference model for WFMS proposed by the WfMC.

TABLE I
THE WORKFLOW PATTERNS

PATTERN TYPE	PATTERN NAME (*SYNONYMS*)	DEFINITION
Basic Control Flow	Sequence (*Sequential routing, serial routing*)	An activity is enabled after the completion of another activity.
	Parallel Split (*AND-split, parallel routing, fork*)	A point in the workflow where a single thread of control splits into multiple threads of control which can be executed in parallel, thus allowing activities to be executed simultaneously or in any order.
	Synchronization (*AND-join, rendezvous, synchronizer*)	A point in the workflow where multiple parallel subprocesses/activities converge into one single thread of control, thus synchronizing multiple threads.
	Exclusive Choice (*XOR-split, conditional routing, switch, decision*)	A point in the workflow where, based on a condition, one of several branches is chosen.
	Simple Merge (*XOR-join, asynchronous join, merge*)	A point in the workflow where two or more alternative branches merge without synchronization.
Advanced Branching and Synchronization	Multi-choice (*Conditional routing, selection, OR-split*)	A point in the workflow where, based on a condition, a number of branches are chosen.
	Synchronizing Merge (*Synchronizing join, OR-join*)	A point in the workflow where multiple paths converge into one single thread.
	Multi-merge	A point in a workflow where two or more branches reconverge without synchronization. If more than one branch gets activated the activity following the merge is started for every activation of every incoming branch.
	Discriminator (*N/M or partial join*)	A point in a workflow that waits for a number of the incoming branches to complete before activating the subsequent activity; then it waits for the remaining branches to complete and "ignores" them. Then it resets itself.
Structural	Arbitrary Cycles (*Loop, iteration, cycle*)	A point in a workflow where one or more activities can be done repeatedly.
	Implicit Termination	A given sub-workflow should be terminated when there is nothing else to be done.
Multiple Instances	Multiple Instances without synchronization (*Multi-threading without synchronization, spawn off facility*)	Multiple instances of an activity can be created with no need to synchronize them.
	Multiple Instances with a priori design time knowledge	An activity is enabled a number of times known at design time. Once all instances are completed some other activity needs to be started.
	Multiple Instances with a priori run-time knowledge	An activity is enabled a number of times known at run time. Once all instances are completed some other activity needs to be started.
	Multiple Instances without a priori run-time knowledge	An activity is enabled a number of times known neither at design time nor at run-time. Once all instances are completed some other activity needs to be started.
State-based	Deferred Choice (*External choice, implicit choice, deferred XOR-split*)	It is similar to the exclusive choice but the choice is not made explicitly and the run-time environment decides what branch to take.
	Interleaved Parallel Routing (*Unordered sequence*)	A set of activities is executed in an arbitrary order decided at run-time; no two activities are active at the same time.
	Milestone (*Test arc, deadline, state condition, withdraw message*)	The enabling of an activity depends on the workflow being in a given state, i.e. the activity is only enabled if a certain milestone has been reached which did not expire yet.
Cancellation	Cancel Activity (*Withdraw activity*)	An enabled activity is disabled, i.e. a thread waiting for the execution of an activity is removed.
	Cancel Case (*Withdraw case*)	A workflow instance is removed completely.

As described above, the Process Definition component provides the process designer with a workflow language able to specify a workflow schema which can be successively instantiated by means of the Enactment Service based on a workflow API and on one or more workflow engines. The workflow definition language is to be expressive and powerful to specify complex workflows from several perspectives: control-flow, data-flow, resource and operational [25].

The control-flow perspective describes activities and their execution ordering through different constructors, which permit to control the flow of execution, and provides an essential insight into the effectiveness of a workflow specification. The data flow perspective rests on control-flow perspective, while the resource and operational perspectives are ancillary.

An expressive and powerful set of control-flow constructs for the specification of workflow schemas (WF Schemas) is the set of the Workflow Patterns proposed by van der Aalst [25]. In Table I the Workflow Patterns, identified by examining the most known contemporary workflow management systems, are enumerated along with their synonyms and a brief definition.

An example WF Schema based on the WF Patterns and drawn by using YAWL [24] is given in Figure 2. After the task A is carried out (sequence pattern), the tasks B, C, and D are executed in parallel (parallel split pattern). When B or C complete (synchronizing merge pattern), the task E is executed an arbitrary number of times (arbitrary cycles pattern). When D completes (sequence pattern), either the task F or the task G (exclusive choice pattern) is executed. When either F or G complete (simple merge pattern), depending on the precedent choice, the task H is executed (sequence pattern). When the iterative execution of E completes and also H terminates (synchronization pattern), the task I is executed and, after its completion (sequence pattern), the workflow terminates.

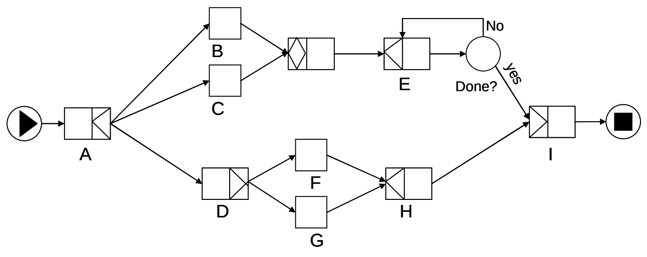


Fig. 2. An example WF Schema based on the WF Patterns.

A generic Workflow Enactment Service (WFES) receives from the user the indication about which workflow is to be enacted (WF Type param) and the Case Activation Record (CAR) for the specific workflow instance; then, on the basis of the WF Schema corresponding to the selected WF Type, the WFES enacts the workflow by means of a specific WF Engine. If the WF Engine is of the distributed type the user indicates also a set of params for specifying some requirements related to the distribution of control, computation and/or data during the workflow enactment (Figure 3).

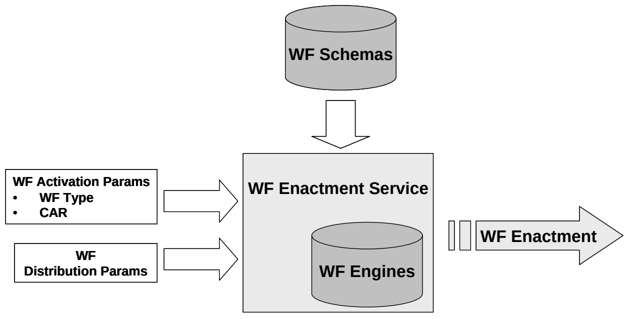


Fig. 3. An A Workflow Enactment Service.

More in details, in order to enact a workflow the WFES selects a suitable WF Engine, from the WF Engines repository, on the basis of the schema of the workflow to be enacted. If the required WF Engine is not available the WFES creates it. The creation is driven by the WF Schema which is used to properly instantiate a Workflow Enactment Framework so building a WF Engine able to enact that specific WF Schema. In building the specific WF Engine the distribution params specified by the user are also considered. Finally, the WF Engine will enact the workflow on the basis of the given CAR and the possible distribution params (Figure 4). The created WF Engine is stored in the WF Engines repository so that it can be reused for enacting future workflows of the same type.

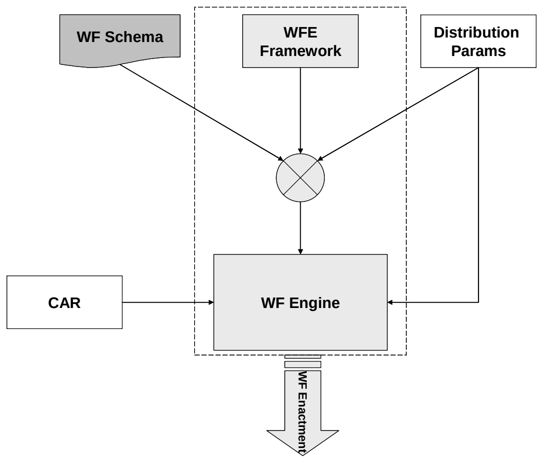


Fig. 4. The construction of a WF Engine.

A. The proposed agent-based approach

In our approach for the distributed workflow enactment a WF Engine is a MAS built by properly instantiating the Agent-based Workflow Enactment Framework (AWEF) on the basis of a WF Schema defined by using the WF Patterns. In particular, a WF Engine consists of tree different agent types:

- *EnacterAgent*, which represents the interface between the MAS constituting the WF Engine and the Workflow

Enactment Service and is responsible for the activation and monitoring of the workflow.

- *ManagerAgent*, which is responsible of the execution and control of the workflow. A single *ManagerAgent* allows for flat workflow management whereas a hierarchical structure of *ManagerAgent*s, formed according to the parent/child model, allows for a hierarchical workflow management. The behavior of a *ManagerAgent* is defined on the basis of the WF Schema it has to enact.
- *TaskAgent*, which is responsible for the execution of internal tasks and/or for the wrapping of external tasks or services. The behavior of a *TaskAgent* is defined on the basis of the activities composing the task it has to carry out.

A WF Engine is, therefore, a MAS with a hierarchical organizational structure in which the control of the workflow execution is hierarchically distributed between the *ManagerAgent*s and the computation is distributed among the *TaskAgent*s.

A WF Schema can be specified by using YAWL [24] which is based on the WF Patterns and offers also an XML based representation of the WF Schema

The design and the implementation of the AWEF, on which the WF Engines are based, are presented in details in sections 3 and 4.

B. *Other related approaches*

In the literature it is possible to find different proposals of distributed workflow enactment mechanisms based on the Agent paradigm and technologies which aim to support more flexible, dynamic and adaptive workflow from the process, resource and activity perspective [8].

Such approaches differ from each other in the supported dimensions of distribution (computation, control and data), in the adopted coordination model (control-driven, data-driven) and in the exploited MAS organizational structure (hierarchical, peer-to-peer).

In [10,23,21] the authors present an agent-based workflow engine centered on a hierarchical organizational structure in which a *ProcessAgent* executes a workflow instance by requesting the execution of the tasks composing the workflow to a set of *ResourceAgent*s. *ResourceAgent*s can be seen as representing web services and can be dynamically discovered and allocated to a *ProcessAgent* by a *ResourceBrokerAgent*. In this control-driven approach the control about the state of the workflow execution is hierarchically distributed between the *ProcessAgent*s and the computation whereas data are distributed among the *ResourceAgent*s which are responsible of the task execution.

In [1] the authors propose a software environment to dynamically generate agent-based workflow engines. A workflow engine is generated by a compiler that translates an XPDL workflow definition to a MAS ready to be executed in the Hermes middleware. The translation process is a two step procedure. In the first step the user-level workflow definition is mapped to an Agent Level Workflow (ALW) specification. In the second step, the compiler concretely generates agents, called Workflow Executors, from the ALW specification by plugging the implementation of the required workflow activities, that are available in a repository, into “empty” agents (skeletons). A workflow engine is, therefore, a MAS having a peer-to-peer organizational structure in which the workflow execution is driven by the interactions among the Workflow Executors.

A similar approach can be found in [22] which presents a methodology for translating a workflow specification into a MAS architecture specifying formalized rules for modeling agents’ behaviors. The MAS is not generated automatically by a compiler like in [1] but by the developer adopting a tool called Agent Developer Studio (ADS).

In [7,8,9] the authors present an agent-based approach for enacting BPEL4WS (Business Process Execution Language for Web Services) [6] workflow specifications. BPEL4WS is an XML-based language that allows for the specification of workflows where the activities are defined by Web service invocations. The proposed distributed enactment mechanisms combine data-centered and control-centered coordination mechanisms. Data are managed via a shared XML repository while the control of the workflow activities is driven by asynchronous messages exchanged between the agents that enact the workflow. The message exchange pattern for the control messages is derived from a Colored Petri Net model of the workflow. The agents’ behaviors are configured and instantiated at run time on the basis of the BPEL4WS specification of the specific workflow to be enacted. The organizational MAS structure is based on a *RequestorAgent* that orchestrates a set of Distributed Workflow Agents according to the workflow specification. The system has been implemented in JADE.

Another agent-based approach for enacting workflows specified in BPEL4WS is proposed in [14]. The novelty of the approach is that the enactment of the workflows is carried out by peer agents that can be associated with web services. The control flow is coded in an interaction protocol that is not defined at the development time like in [1,22] but which is passed at run time between the agents together with the messages so informing each agent what to do next to keep the workflow executing.

Another peer-to-peer agent-based enactment approach is presented in [29]. In this approach the workflow to be enacted is decomposed into a set of interrelated task partitions. Each task partition represents a service and its position, i.e., the interaction and dependency with the other services in the process. Then, each task partition is distributed to an agent which represents a service provider offering a service required by the specific workflow instance. Each agent autonomously manages the enactment of the represented service and the interactions between this service and the others only on the basis of the assigned task partition; agents are not conscious of the whole process in which they are involved. Such adopted coordination model is known as a choreography coordination model.

III. The design of AWEF

The design of AWEF was carried out by exploiting an agent-oriented development process [11] in which the requirements capture phase is supported by the Tropos methodology [5], the analysis and design phases are supported by the Gaia methodology [27] and the detailed design phase is supported by the Agent-UML [2] and the Distilled StateCharts (DSC) [12].

The requirements, captured using the Tropos goal-oriented approach, were reported in a Requirements Statements document. On the basis of the requirements the following key roles were identified:

– *Enacter*, which manages the activation and monitoring of workflows and represents the interface between the WF Engine and the Workflow Enactment Service;
– *Manager*, which manages the execution and control of workflows;
– *Executor*, which executes the internal workflow tasks;
– *Wrapper*, which interacts with the external tasks or services.

Each of these roles was fully described by using a Role Schema according to the Gaia methodology. The protocols associated with each role were identified and documented by an Interactions Model. Then, the identified Roles were aggregated into Agent Types also specifying the agent types hierarchy (Agent Model); the main services required to realize each role were specified (Services Model) and the relationships of communication between the Agent Types documented (Acquaintance Model).

The identified Agent Types are:

– *EnacterAgent*, which derives from the *Enacter* role.
– *ManagerAgent*, which derives from the *Manager* role. A single *ManagerAgent* allows for flat workflow management whereas a hierarchical structure of *ManagerAgent*s, formed according to the parent/child model, allows for a hierarchical workflow management.
– *TaskAgent*, which derives from both the *Executor* and *Wrapper* role.

The detailed design phase allowed for obtaining a detailed specification of the behaviors of the Agent Types which have been defined in the Agent Model. The work products of this phase were the Agent Interactions Model and the Agent Behaviors Model. The former consists of a set of Agent-UML interaction diagrams [2] which thoroughly specify the patterns of interaction between the Agent Types; the Agent Behaviors Model specifies the dynamic behavior of each Agent Type by means of the Distilled StateCharts (DSC) formalism [12].

The main interaction patterns documented by the Agent Interactions Model are:

– *EnacterAgent/ManagerAgent*, which is enabled by the Enacter/Manager Interaction Protocol (EMIP);
– *ManagerAgent/ManagerAgent*, which is enabled by the Manager(parent)/Manager(child) Interaction Protocol (MMIP);
– *ManagerAgent/TaskAgent*, which is enabled by the Manager/Task Interaction Protocol (MTIP).

In the Agent Behaviors Model the basic behaviors of the *EnacterAgent*, *ManagerAgent* and *TaskAgent* are defined. In particular, the defined *ManagerAgent* behavior (or *ManagerBehavior*) is composed of:

– An InitialPseudoActivity, which represents the starting point of the workflow execution in the WF Schema.
– One or more FinalPseudoActivity, which represent points in the WF Schema at which the workflow or a part of it ends. A FinalPseudoActivity uses a parent *ManagerAgent* for notification purposes.
– One or more *WFPattern*, which represent the control-flow activities. A *WFPattern*, which can be any of the available WF Patterns [25] (sequence, and-split, and-join, xor-split, xor-join, or-split, multi-merge, discriminator, loop, multiple instances, deferred choice, milestone, etc) uses one or more *TaskAgent*s and one or more child *ManagerAgent*s for activation purposes and a parent *ManagerAgent* for notification purposes.

In order to model a WF Schema, InitialPseudoActivity, FinalPseudoActivity, and *WFPattern* are linked through source/target control-flow associations.

Figure 5 shows a Statecharts-based representation [15] of the *ManagerBehavior*.

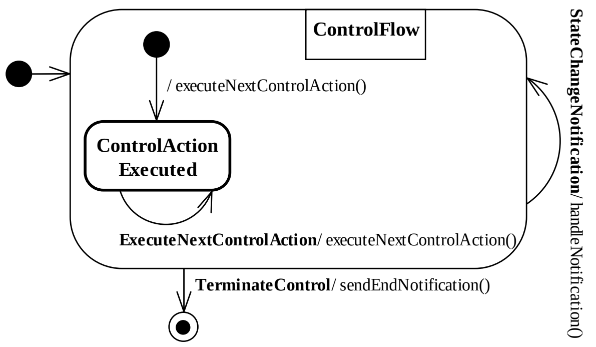

ControlFlow
/ executeNextControlAction()
ControlAction Executed
ExecuteNextControlAction/ executeNextControlAction()
TerminateControl/ sendEndNotification()
StateChangeNotification/ handleNotification()

Fig. 5. The generic behavior of a *ManagerAgent*.

According to the WF Schema to enact, the *ManagerAgent* enters the *ControlFlow* superstate executing the *executeFirstControlAction()* method. In this superstate, every times that an *ExecuteNextControlEvent* is received the *ManagerAgent* executes the next control-flow action by invoking the *executeNextControlAction()* method which fetches the next *WFPattern* and executes it. Upon completion of a *WFPattern* execution, two events are generated: (i) *ExecuteNextControlAction* which allows the *ManagerBehavior* to invoke the *executeNextControlAction()* method; (ii) *StateChangeNotification* which allows notifying the upper-level *ManagerAgent* or the *EnacterAgent* about the control-flow state change of the workflow. If there are no more *WFPatterns* to execute, the *TerminateControl* event is generated which drives the termination of the *ManagerBehavior* and the transmission of the related *EndNotification* to the upper-level *ManagerAgent* or to the *EnacterAgent*. A *WFPattern* execution can involve: (i) the detection of the completion of a task through the reception of a FIPA ACL message which can also carry the data produced by the completed task; (ii) the creation and/or activation of *TaskAgent*s or *ManagerAgent*s.

The interaction diagrams composing the Agent Interactions Model and the behavioral specifications of the Agent Behaviors Model are to be intended as basic schemas that will be coded into the basic classes of AWEF. A WF Engine will be obtained by instantiating such basic classes according to the schema of the workflow to be enacted and using the concrete implementations of the tasks required for the workflow execution.

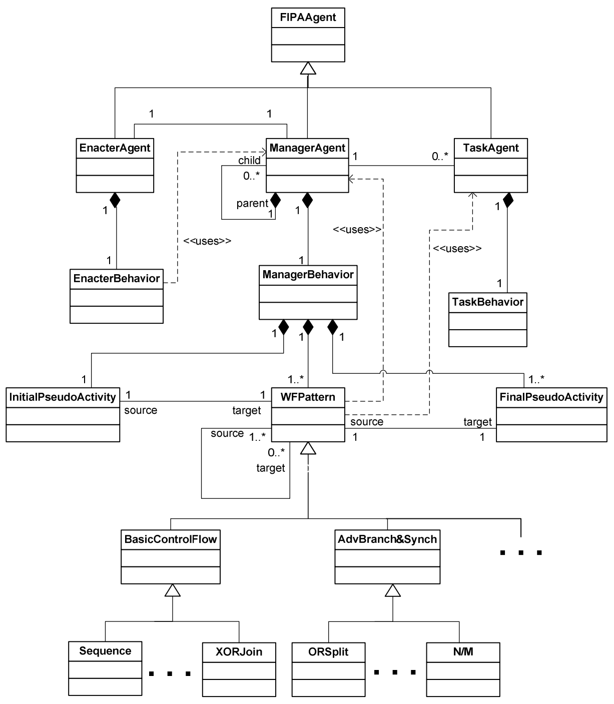


Fig. 6. Class diagram of the AWEF Framework.

In Figure 6 the classes which compose AWEF are reported. In particular, AWEF provides the base agents for workflow enactment (*EnacterAgent*, *ManagerAgent* and *TaskAgent*), their interaction protocols and a set of control-flow classes which are associated to the behavior of the *ManagerAgent* and implement the WF Patterns.

IV. The JADE-based implementation of AWEF

The JADE-based classes of AWEF were straightforwardly derived from the class diagram reported in Figure 6. In particular:

– *EnacterAgent*, *ManagerAgent* and *TaskAgent* extend the *Agent* class of JADE [16];

– *EnacterBehavior*, *TaskBehavior* and *WFPattern* extend *Behaviour* class of JADE which represents a generic behavior terminating when the end-of-activity condition is met;

– *ManagerBehavior* extends *FSMBehaviour* class of JADE which models a complex task whose sub-tasks correspond to the activities performed in the states of a finite state machine. In particular, the states of *ManagerBehavior* correspond to the control-flow states of the workflow (or sub-workflow) that the *ManagerAgent* is controlling; each state is associated to a *WFPattern* which is activated when the state becomes active. EMIP, MMIP, and MTIP are appositely defined through sequences of ACL messages instances of the *ACLMessage* class of JADE.

In the following subsection a hierarchical WF Engine based on AWEF and capable of enacting the WF Schema reported in Figure 2 is presented.

A. A hierarchical WF Engine based on AWEF

The hierarchical workflow management is enabled by a set of *ManagerAgent*s each of which embodies a sub-schema of a WF Schema according to a hierarchical model. With reference to the WF Schema of Figure 2, the WF Engine able to enact such a WF Schema is obtained through:

1. The partitioning of the WF Schema into a set of hierarchically arranged workflow schemas: WF Schema 1, WF Schema 1.1, WF Schema 1.2 (see Figure 7);

2. The instantiation of AWEF with respect to the obtained workflow schemas (see Figure 8 for the resulting class diagram).

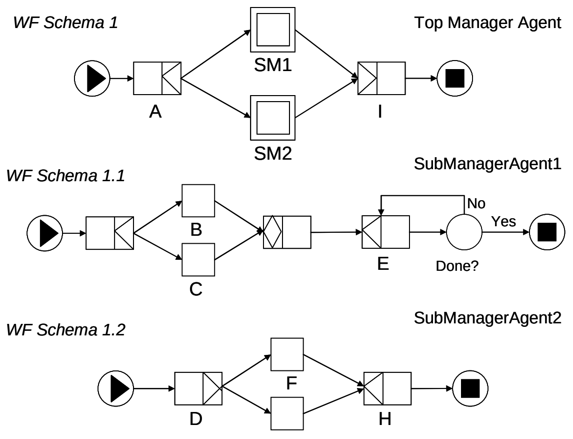


Fig. 7. Partitioned WF Schema.

With reference to Figure 8a the *EnacterAgent* is linked to the top-level *ManagerAgent* which is, in turn, linked to the *TaskAgent*s related to the WF Schema 1, and to the *SubManagerAgents1*and *SubManagerAgent*s*2* which control the WF Schemas 1.1 and 1.2 respectively. Each *SubManagerAgent* is, in turn, linked to the *TaskAgent*s associated to its schema.

With reference to Figures 8b-d each *ManagerBehavior* is obtained by translating its associated WF Schema in a set of classes consisting of one *InitialPseudoActivity*, one or more *FinalPseudoActivity*, and one or more *WFPattern* which are appositely interconnected.

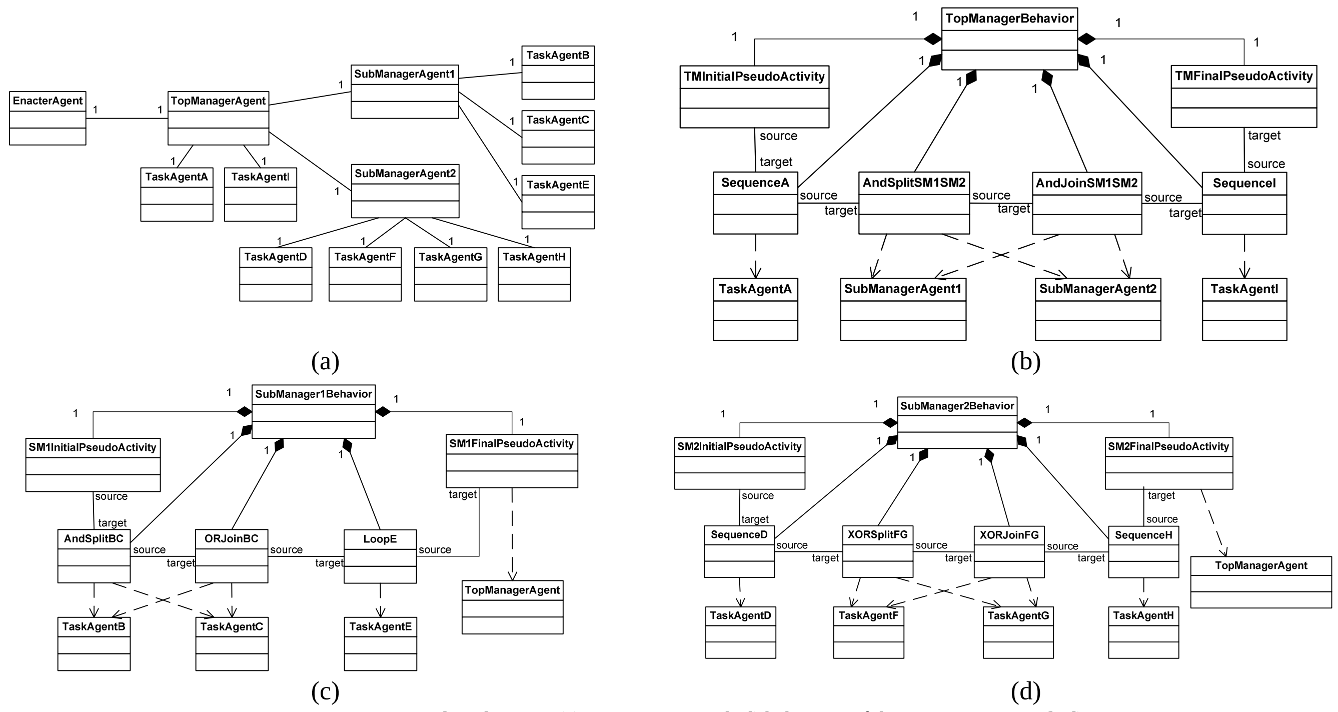


Fig. 8. WF Engine class diagram: (a) MAS structure; (b-d) behaviors of the *ManagerAgent*s (b-d)

V. Conclusions

This paper has described an Agent-based Workflow Enactment Framework (AWEF) which can be instantiated on the basis of a WF Schema for obtaining a specific WF Engine which mainly consists of a hierarchy of *ManagerAgent*s. Each *ManagerAgent* has in charge the enactment of a sub-schema of the WF Schema used for the instantiation of AWEF and exploits a set of *TaskAgent*s for the execution of the specific workflow tasks associated to its sub-schema. This MAS organization allows for the hierarchical distribution of the workflow execution control between the *ManagerAgent*s and for the distribution of the computation among the *TaskAgent*s. Due to these features AWEF constitutes a basic component for the construction of more flexible, efficient, and robust Workflow Enactment Services.

The JADE-based implementation of AWEF has been applied to the development of a workflow system for the monitoring of distributed agro-industrial productive processes. The developed workflow system is a component of a larger system which was built in the context of the M.ENTE (Management of integrated ENTErprise) project which aims at developing a pervasive system for the control and management of productive, organizational, and business processes of companies working in the agro-alimentary industry of Calabria. The current experimentation of the system provides support to a consortium of agro-industrial greenhouses.

Efforts are currently underway to develop an enactment service which is able to automatically instantiate AWEF on the basis of WF Schemas defined in YAWL.

Acknowledgment

This work was partially supported by the Italian Ministry of Education, University and Research in the framework of the M.ENTE (Management of integrated ENTErprise) research project PON (N°12970-Mis.1.3). The authors also wish to thank Serena Martino and Giuseppe Agapito for their contribution to the JADE-based implementation of the AWEF framework.

References

[1] E. Bartocci, F. Corradini, E. Merelli, Enacting proactive workflows engine in e-Science, In proceedings of the 6th International Conference on Computational Science (ICCS 2006), University of Reading, UK, May 28-31, 2006, pp. 1012-1015, volume 3993/2006, Springer-Verlag, Berlin.
[2] B. Bauer, J.P. Muller, and J. Odell. Agent UML: A Formalism for Specifying Multiagent Interaction. In Paolo Ciancarini and Michael Wooldridge, editors, *Agent-Oriented Software Engineering*, pages 91-103. Springer-Verlag, Berlin, 2001.
[3] F. Bellifemine, A. Poggi, and G. Rimassa, Developing multi-agent systems with a FIPA-compliant agent framework, *Software Practice and Experience,* 31, pp. 103-128, 2001.
[4] U. M. Borghoff, J. H. Schlichter. Computer-Supported Cooperative Work: Introduction to Distributed Applications. Springer-Verlag. 2000.
[5] P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, and A. Perini. TROPOS: An Agent-Oriented Software Development Methodology, *Journal of Autonomous Agents and Multi-Agent Systems*, 8(3):203-236, 2004.
[6] Business Process Execution Language for Web Services version 1.1. http://www-128.ibm.com/developerworks/library/specification/ws-bpel/.
[7] P. Buhler, J.M. Vidal, Towards AdaptiveWorkflow Enactment Using Multiagent Systems, *Journal of Information Technology and Management*, vol. 6, pp. 61–87, 2005, Springer-Verlag, Berlin.

[8] P. Buhler, J.M. Vidal, Enacting BPEL4WS specified workflows with multiagent systems, In Proceedings of the Workshop on Web Services and Agent-Based Engineering, 2004.

[9] P. Buhler, J.M. Vidal and H. Verhagen, Adaptive Workflow = web services + agents, In Proceedings of the First International Conference on Web Services, 131-137, 2003.

[10] L. Ehrler, M. Fleurke, M. A. Purvis, and B.T.R. Savarimuthu, Agent-Based Workflow Management Systems(Wfmss): JBees - A Distributed and Adaptive WFMS with Monitoring and Controlling Capabilities, *Journal of Information Systems and e-Business Management*, Volume 4, Issue 1, Jan 2006, Pages 5-23, Springer-Verlag, Berlin.

[11] G. Fortino, A. Garro, and W. Russo, An Integrated Approach for the Development and Validation of Multi Agent Systems, *Computer Systems Science & Engineering*, 20(4), pp.259-271, Jul. 2005.

[12] G. Fortino, W. Russo, and E. Zimeo, A Statecharts-based Software Development Process for Mobile Agents, *Information and Software Technology*, 46(13), pp. 907-921, Oct. 2004.

[13] Foundation of Intelligent and Physical Agents, http://www.fipa.org.

[14] L. Guo, Y. Chen-Burger, and D. Robertson Dave, Enacting the Distributed Business Workflows Using BPEL4WS on the Multi-Agent Platform. In Proceedings of the Third German Conference on Multi-agent System Technologies (MATES2005), LNAI 3550, pp. 35-47, Koblenz Germany, Springer-Verlag.

[15] D. Harel and E. Gery. Executable Object Modelling with Statecharts. *IEEE Computer*, 30(7): 31-42, 1997.

[16] Java-based Agent Development Environment (JADE), documentation and software at the world wide web: http://jade.tilab.com.

[17] N.R. Jennings, P. Faratin, T.J. Norman, P. O'Brien, and B. Odgers, Autonomous Agents for Business Process Management, *Journal of Applied Artificial Intelligence*, 14(2), pp. 145–189, 2000.

[18] M. Luck, P. McBurney, and C. Preist, A Manifesto for Agent technology: Towards Next Generation Computing, *Autonomous Agents and Multi-Agent Systems*, 9(3), pp. 203-252, 2004.

[19] D.C. Marinescu, Internet-based Workflow Management, John Wiley & Sons, Inc., New York, 2002.

[20] P.D. O'Brien and M.E. Wiegand, Agent-based process management: applying intelligent agents to workflow, *Knowledge Engineering Review*, 13(2), pp. 1-14, 1998.

[21] M. A. Purvis, B.T.R Savarimuthu, and M.K Purvis, A Multi-agent Based Workflow System Embedded with Web Services, In proceedings of the second international workshop on Collaboration Agents: Autonomous Agents for Collaborative Environments (COLA 2004), Beijing, China, September 2004, pp 55-62, IEEE/WIC Press.

[22] M. Repetto, M. Paolucci, A. Boccalatte, A Design Tool to Develop Agent-Based Workflow Management Systems, In Proc. of the 4th AI*IA/TABOO Joint Workshop "From Objects to Agents": Intelligent Systems and Pervasive Computing, 10-11 September 2003, Villasimius, CA, Italy, pp. 100-107, Pitagora Editrice Bologna.

[23] B.T.R. Savarimuthu, M.A. Purvis, M.K. Purvis, and S. Cranefield, Agent-Based Integration of Web Services with Workflow Management Systems, *Information Science Discussion Paper Series*, Number 2005/05, ISSN 1172-6024, University of Otago, Dunedin, New Zealand.

[24] W.M.P. van der Aalst and A.H.M. ter Hofstede. YAWL: Yet Another Workflow Language. *Information Systems*, 30(4):245-275, 2005.

[25] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros, Workflow Patterns, *Distributed and Parallel Databases*, 14(3), pp. 5-51, July 2003.

[26] W.M.P. van der Aalst and K. van Hee, Workflow Management: Models, Methods, and Systems, The MIT Press, Cambridge (MA), 2002.

[27] M. Wooldridge, N. R. Jennings, and D. Kinny. The Gaia methodology for agent-oriented analysis and design. *Journal of Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.

[28] Workflow Management Coalition, http://www.wfmc.org.

[29] J. Yan, Y. Yang, R. Kowalczyk, and X. T. Nguyen, A service workflow management framework based on peer-to-peer and agent technologies, In Proc. of the International Workshop on Grid and Peer-to-Peer based Workflows co-hosted with the 5th International Conference on Quality Software, Melbourne, Australia, September 19 -21, 2005.

A Lightweight Architecture for RSS Polling of Arbitrary Web sources

Sergio Bossa, Giacomo Fiumara and Alessandro Provetti
Dip. di Fisica, Università degli Studi di Messina
Sal. Sperone 31, I-98166 Messina, Italy
sergio.bossa@gmail.com, {fiumara,ale}@unime.it

***Abstract*— We describe a new Web service architecture designed to make it possible to collect data from traditional plain HTML Web sites, aggregate and serve them in more advanced formats, e.g. as RSS feeds. To locate the relevant data in the plain HTML pages, the architecture requires the insertion of some meta tags in the commented text. Hence, the extra markup remains totally transparent to users and programs. Such annotated HTML documents are then routinely pulled by our Web service, which then aggregates the data and serves them over several channels, e.g. RSS 1.0 or 2.0. Also, a REST-style Web Service allows users to submit XQuery queries to the feeds database. Finally, we discuss scalability issues w.r.t. polling frequencies.**

I. Introduction

This article describes a new, experimental architecture for automated data collection and RSS delivery of data from traditional HTML Web sites. Our solution requires minimal and totally transparent changes on their HTML pages. The data of interest will be routinely *polled* from the actual sources by standard HTTP querying. Subsequently, the so-created Web service can be queried with REST-style sessions that extract the aggregated data at their wish. As a result, we provide a complete layout for the implementation of RSS Web services that interact with the traditional Web in an almost seamless way.

Even though this research project is only at the beginning, and only a proof-of-concept implementation is available, we believe that there is room for the application of this type of approach to bridging Web services and the traditional Web. Let us discuss why. Today we find on the Web several interesting, popular news sites that consist, essentially, of plain old HTML pages. Even though the content is continuously updated, the site layout and organization is not changing much. Several advanced techniques for news broadcasting and syndication are now available, the main one being RSS feeds, yet it seems that a large set of relevant news sources will carry on *by inertia,* with their existing Web architecture. Our architecture enables extracting the relevant data from plain HTML and makes it available to the contemporary Web service techniques.

Indeed, today Web portals are publishing, along with traditional HTML pages, RSS documents, mostly known as RSS feeds [1], [2].

Inasmuch as HTML is aimed at content visualization for end user experience, RSS is an XML format aimed at capturing channels of data items, thus enabling automated data processing. RSS today is used mainly for content syndication; it organizes the semantics in a *channel* element, containing overall information regarding the resource, and a set of *item* elements, each containing logically related pieces of information. Moreover, every channel or item contains a *title* element, a *link* element and a *description* element.

Even though it has been developed for syndication purposes, RSS can be applied to realize sophisticated forms of content manipulation, like aggregation or advanced querying. Using RSS feeds is indeed simple: Web portals must publish, together with HTML documents, the related feeds. Users can then *consume* these feeds by a particular client, called RSS aggregator, by which they can read, query or *aggregate* feeds.

However, this simple process has some limitations: Web masters have to create their RSS feeds by some RSS generation tool, which are often proprietary and may limit interoperability. Moreover, users may not be able to view older feeds, nor to query feeds *on the fly,* directly on the server.

The architecture described here[1] overcomes these limitations by proposing a *pull-based* Web service to generate, store, aggregate and query contents using RSS standards. With this application it is possible to:

- Automatically and dynamically generate RSS feeds starting from HTML Web pages
- Store them in chronological order
- Query and aggregate them thanks to REST [4], [5] Web services acting as software agents

Clearly, there are scalability issues involved in our architecture, and the pulling policy for each site must be carefully considered. Section VI-B below describes a common structure for pulling policies.

II. Adding meta-tags to existing HTML pages

HTML documents contain a mixture of information to be published, i.e., meaningful to humans, and of directives, in the form of tags, for graphical formatting, i.e. intended for browsers interpretation. Moreover, since the HTML format is designed for visualization purposes only, its tags do not allow sophisticated machine processing of the information contained therein.

[1]The architecture was first outlined in the first author's graduation project [3].

Among other things, one factor preventing the spread of the Semantic Web is the complexity of extracting, from existing, heterogeneous HTML documents machine-readable information. Although our project addresses only a fraction of the Semantic Web vision, our management of HTML documents needs some technique to locate and extract some valuable and meaningful content.

Therefore, we define a set of annotations in form of meta-tags, which can be inserted inside an HTML document in order give it semantic structure and highlight informational content. In our application, meta-tags are used as annotations, to describe and mark all interesting information, in order to help in the extraction and so-called *XML-ization* phases. The set of meta-tags we defined (and recognizable by our application) is listed in Table I below. The meta-tags are enclosed in HTML comment tags, so they remain transparent to Web browsers and do not alter the original HTML structure of the document.

The conceptual model of the meta-tags described above is rather straightforward and remains orthogonal to the object tags found in the page.

A. Meta tags vs. dynamic XSLT transformations

An obvious alternative to our approach to the treatment of existing HTML structures is that of applying, after the polling phase, some clever XSLT transformation [6] to the HTML file. It should be considered, however, that applying such type of XSLT transformations is possible (or at least greatly facilitated) only when the [X]HTML document is well-formed. This, regrettably, seems rather unrealistic to us, exp. for old documents. Viceversa, our solution relieves the webmasters from any time-consuming translation of her HTML documents into well-formed XHTML ones, which would then make a subsequent XSLT transformation successful.

III. STRUCTURE OF THE XML OUTPUT

Once HTML documents are processed by our application, annotated semantic structures are extracted and organized into a simple XML format which will be stored and used as a starting point for document querying and transformation. This XML format has been simply called *XMLData*. This *neutral* format has also been introduced in order to avoid storing the same information in both RSS 1.0 and 2.0 formats. Indeed, we found more economic for our application to create RSS feeds *on the fly* rather than store them. This approach is also more flexible as the support of new syndication formats (see for example, the Atom format) does not require the re-design of the lower levels of the application (see further). The structure of the XML output resembles the structure of meta-tags previously defined and the RSS XML structure, in order to facilitate transformations from the former to the latter. It is defined by the following XML Schema Definition [7]:

```
<?xml version="1.0" encoding="UTF-8"?>

<xsd:schema
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="unqualified">

<xsd:complexType name="imageType">
  <xsd:all>
    <xsd:element name="title" type="xsd:string"/>
    <xsd:element name="link" type="xsd:anyUri"/>
    <xsd:element name="url" type="xsd:anyUri"/>
  </xsd:all>
</xsd:complexType>

<xsd:complexType name="extensionsType">
  <xsd:sequence>
    <xsd:any namespace="##any"
     processContents="skip"
        minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="channelType">
  <xsd:all>
    <xsd:element name="title" type="xsd:string"/>
    <xsd:element name="link" type="xsd:anyUri"/>
    <xsd:element name="description"
     type="xsd:string"/>
    <xsd:element name="image" type="imageType"
        minOccurs="0" maxOccurs="1"/>
    <xsd:element name="extensions"
     type="extensionsType"
     minOccurs="0" maxOccurs="1"/>
  </xsd:all>
</xsd:complexType>

<xsd:complexType name="itemType">
  <xsd:all>
    <xsd:element name="title" type="xsd:string"/>
    <xsd:element name="link" type="xsd:anyUri"/>
    <xsd:element name="description"
     type="xsd:string"/>
    <xsd:element name="image" type="imageType"
        minOccurs="0" maxOccurs="1"/>
    <xsd:element name="extensions"
     type="extensionsType"
        minOccurs="0" maxOccurs="1"/>
  </xsd:all>
  <xsd:attribute name="index" type="xsd:integer"
   use="required"/>
  <xsd:attribute name="id" type="xsd:string"
   use="required"/>
</xsd:complexType>

<xsd:complexType name="resourceType">
  <xsd:sequence>
    <xsd:element
     name="channel" type="channelType"/>
    <xsd:element
     name="item" type="itemType"
        minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="url" type="xsd:anyUri"
   use="required"/>
  <xsd:attribute name="rssId"
   type="xsd:string" use="required"/>
  <xsd:attribute name="timestamp"
   type="xsd:dateTime" use="required"/>
</xsd:complexType>

<xsd:element name="resource" type="resourceType">
  <xsd:key name="itemId">
    <xsd:selector xpath="item"/>
    <xsd:field xpath="@id"/>
  </xsd:key>
  <xsd:key name="itemIndex">
    <xsd:selector xpath="item"/>
    <xsd:field xpath="@index"/>
  </xsd:key>
```

Meta-tag	Description
<channel:title> ...</channel:title>	Channel title
<channel:description > ...< /channel:description>	Channel description
<channel:image url=" link=" title=" />	URL, link and title of an image associated to the channel
<channel:extension uri=" prefix="> ...</channel:extension>	Channel extension (e.g., publication date)
<item:link index="> ...</item:link >	item link
<item:description index="> ...</item:description >	item description
<item:extension uri=" prefix="> ...</item:extension >	item extension (e.g., item publication date)

TABLE I
THE SET OF META-TAGS

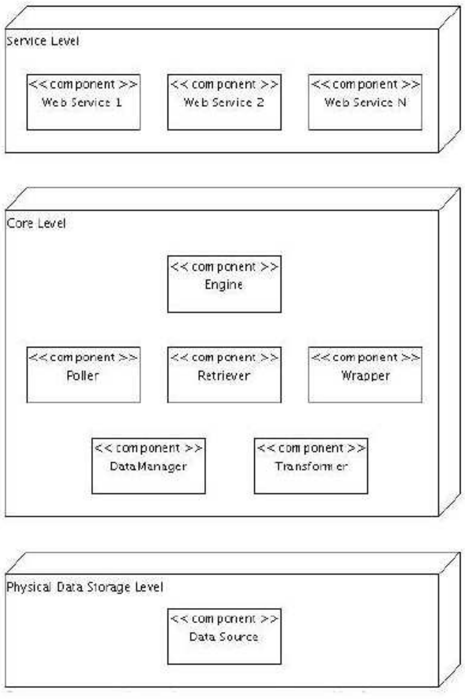

Service Level
<< component >> Web Service 1
<< component >> Web Service 2
<< component >> Web Service N
Core Level
<< component >> Engine
<< component >> Poller
<< component >> Retriever
<< component >> Wrapper
<< component >> DataManager
<< component >> Transformer
Physical Data Storage Level
<< component >> Data Source

Fig. 1. The general schema

```
  </xsd:element>
</xsd:schema>
```

IV. THE OVERALL APPLICATION ARCHITECTURE

Figure 1 shows the overall architecture of the application. Our application is based on a modular structure, for maximizing the flexibility and the extensibility of configuration. Three levels can be distinguished:

- Physical Data Storage Level It is the lowermost level, which stores resources, and provides a means for retrieving and querying them. It can be implemented in various ways, using also established technologies like relational or XML database[8].
- Core Level. It holds the core part of the entire architecture, including the software components which implement the logic of information management and processing; each component can be implemented using different strategies or algorithms, and plugged into the system without affecting other components, i.e., by simply tuning the application configuration files.
- Service Level. It is the highest level, interacting with Web clients by means of REST Web services.

A more detailed explanation follows, starting from the Core level, the foundation over which our application is based.

A. The Core Level

The Core level is composed by several components defining how the application i) retrieves HTML resources, ii) processes to extract information about channeling, iii) manages this new piece of information and finally iv)transforms and prepares it for client consumption:

- **engine:** the code that routinely invokes the Retriever and thus the whole polling process.
- **Poller**: it monitors changes in a set of HTML resources configured in a particular file, using some polling policy (see next section). Moreover, the poller has the important task of coordinating other components in the retrieving, extraction, and storing phases.
- **Retriever**: when invoked by the Poller, it captures the Web resource from its URL and makes it available to other components.
- **Wrapper**: it takes care of extracting the annotated semantic structures from the retrieved HTML resources, wrapping them in a new one, that is, assembling the extracted structures in a fresh, pure XML format, containing the desired informational content: the previously-described *XMLData* format. So, this component must produce a well formed XML document, ready to be stored by the Physical Data Storage level.
- **DataManager**: it acts as a gateway to the Physical Data Storage level, taking care of managing information in the form of the new XML documents previously created, storing them and permitting client components to query their contents.
- **Transformer**: it finally takes care of transforming the stored XML documents into the RSS format requested by clients, using XSLT transformations.

Typical parameters of this level can be changed simply modifying the corresponding parameters which are listed in some configuration files, in XML format. The configuration file of the Engine Component, for example, allows to set the type of polling policy of the Web resources. Currently, the choice is between *flat*, i.e, constant over time, or *smart*, i.e., depending on the recent rate of updates. Other parameters are: the type of data manager (currently, the Exist native-XML database together with its connection parameters) and the format of the RSSs sent to Dynamo subscribers (currently RSS1 and RSS2).

B. The Physical Data Storage Level

The Physical Data Storage level can be implemented with various technologies: our choice has been to implement it using a native XML database. This choice allows us to store and manage XML documents produced by the Wrapper software component in their native format, and to use the powerful XQuery language for advanced content querying and aggregation. The native XML database is organized as a set of collections of XML resources, where the nesting of collections is allowed. In our application, we store XML resources as provided by the Wrapper software component, one collection for each resource. Each collection holds the various chronological versions of the resource: so, each collection effectively contains the history of the resource, all its informational content and a changelog.

When a new resource is to be stored, a check is done by the DataManager software component, in order to avoid duplicate resources. Two resources are considered to be different if their informational content changes. More precisely, they are different if changes to titles, links or descriptions of the resource channel or items are detected. Once stored, the resource is chronologically archived and ready for later retrieving and querying.

C. The Service Level

The Service level lets Web clients access the RSS feeds through the use of REST Web Services [9]. REST, an acronym for *Representational State Transfer,* is an architectural style which conceives everything as a resource identified by a URI. In particular, it imposes a restriction about of the URL defining the page info, that, in the REST view, are considered resources. Each resource on the Web, such as a particular part specification file, must have a unique URL (without GET fields after it), that totally represents it.

With respect to the well-known SOAP architecture[2], in REST we never access a method on a service, but rather a resource on the Web, directly using the standard HTTP protocol and its methods, To put it differently, in REST the hypertext linking controls the application state. This feature of REST allows greater simplicity and maximum interoperability with any Web client, either *thick,* like a desktop application, or *thin,* like a Web browser.

D. Accessing REST Web Services and resources

Adhering to the REST architecture and vision, everything is a resource and so any request and any search returns to the client an RSS resource, actually in the format of RSS 1.0 or 2.0, depending on the client choice.

[2]Please refer to [10] for an introduction to SOAP

In our application, these RSS resources are accessed through HTTP requests, using the GET method of HTTP 1.1 protocol; clients can ask for:

- A list of collections of RSS resources, each representing the chronological history of a resource.
- A list of RSS resources contained in a given collection.
- An RSS resource, identified by an index.
- An RSS resource containing only up to a given number of items, starting from the most recent one.
- An RSS resource obtained by querying a collection of resources, searching for keywords in titles, links or descriptions of items.

The GET method, in principle, should not modify the original resource. A detailed description of how REST Web Services and resources are accessed follows.

a) /resources[?type=rssType]: Accesses an RSS resource listing all collections of resources that clients can request and query. The optional *type* parameter identifies the RSS type of the requested resource.

b) /resources/rssId[?type=rssType]: Accesses an RSS resource listing all resources contained in the collection identified by the resource id, the *rssId* URL section. The optional *type* parameter identifies the RSS type of the requested resource.

c) /resources/rssId?index=n & [type=rssType]: Accesses an RSS resource identified by its *rssId* and the *index* parameter, that is the index number into the chronological history: use ”1” for the first resource (the most recent one), ”2” for the second and so on. The optional *type* parameter identifies the RSS type of the requested resource.

d) /resources/rssId?max=n & [type=rssType]: Accesses an RSS resource identified by its *rssId*, containing only up to *max* items. The optional *type* parameter identifies the RSS type of the requested resource.

e) Complex queries: The following query:

```
/resources/rssId?max=n & [type=rssType]
                       & [(title| link | description | desc)=value]
                       & [op=(and| or)]
                       & [(title| link | description| desc)=value]
                       & ...
```

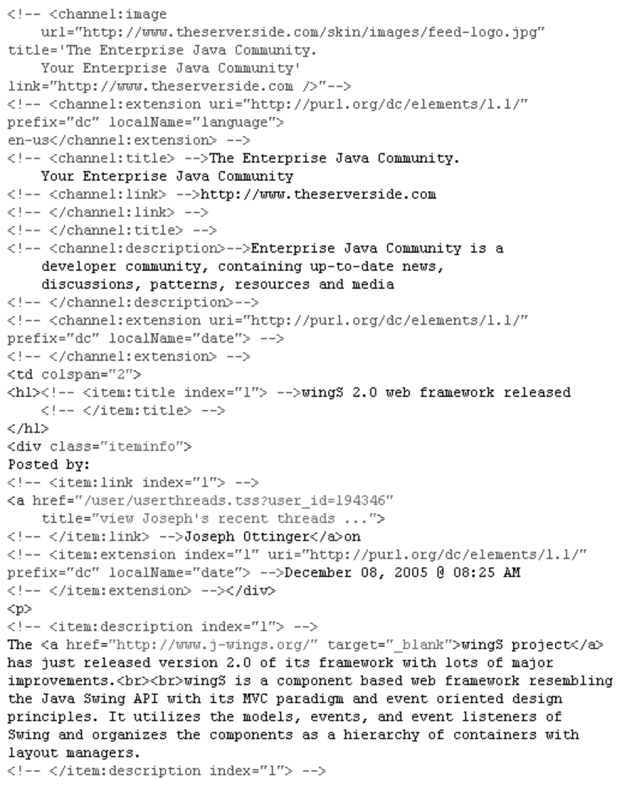
is intended to query all resources identified by the given *rssId*, requesting only up to *max* items and combining, using logical ”and/or” operators, searches for title, link, or description of items. The optional *type* parameter identifies the RSS type of the requested resource.

V. THE APPLICATION AT WORK

To illustrate how our application works we consider a fragment of a HTML document taken from the reference Web site *www.theserverside.com.* After the insertion of the meta-tags, the fragment looks as in Figure 2. Then the fragment is converted in XML format and, if not already present in the database, is stored in the appropriate collection of the database. Upon request from the client, the XML file is extracted and converted into one of the two formats currently supported by our application, that is to say RSS 1.0 or RSS 2.0. For sake of brevity we present here only the RSS2 version of the output (see Figure 3). It should be noted that in order to work properly our application strongly relies upon the insertion of

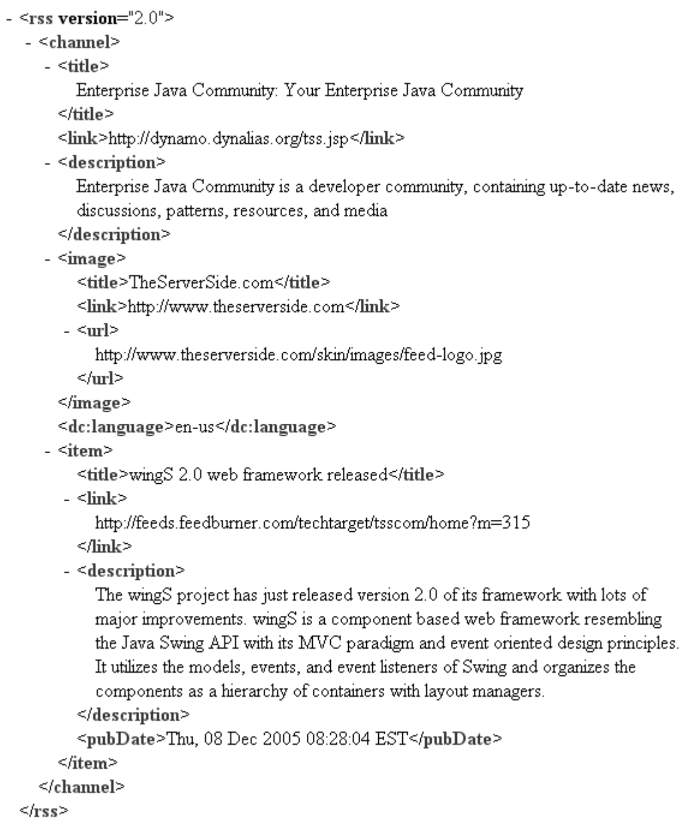
Collection path	Description
/db/resources	Root collection
/db/resources/headlines.rss	Collection holding XML resources related to the headlines.rss resource, that is, its history
/db/resources/headlines.rss/123	XML resource identified by its time-stamp (123)

TABLE II
COLLECTION EXAMPLES



```
<!-- <channel:image
    url="http://www.theserverside.com/skin/images/feed-logo.jpg"
title='The Enterprise Java Community.
    Your Enterprise Java Community'
link="http://www.theserverside.com />"-->
<!-- <channel:extension uri="http://purl.org/dc/elements/1.1/"
prefix="dc" localName="language">
en-us</channel:extension> -->
<!-- <channel:title> -->The Enterprise Java Community.
    Your Enterprise Java Community
<!-- <channel:link> -->http://www.theserverside.com
<!-- </channel:link> -->
<!-- </channel:title> -->
<!-- <channel:description>-->Enterprise Java Community is a
    developer community, containing up-to-date news,
    discussions, patterns, resources and media
<!-- </channel:description>-->
<!-- <channel:extension uri="http://purl.org/dc/elements/1.1/"
prefix="dc" localName="date"> -->
<!-- </channel:extension> -->
<td colspan="2">
<hl><!-- <item:title index="1"> -->wingS 2.0 web framework released
    <!-- </item:title> -->
</hl>
<div class="iteminfo">
Posted by:
<!-- <item:link index="1"> -->
<a href="/user/userthreads.tss?user_id=194346"
    title="view Joseph's recent threads ...">
<!-- </item:link> -->Joseph Ottinger</a>on
<!-- <item:extension index="1" uri="http://purl.org/dc/elements/1.1/"
prefix="dc" localName="date"> -->December 08, 2005 @ 08:25 AM
<!-- </item:extension> --></div>
<p>
<!-- <item:description index="1"> -->
The <a href="http://www.j-wings.org/" target="_blank">wingS project</a>
has just released version 2.0 of its framework with lots of major
improvements.<br><br>wingS is a component based web framework resembling
the Java Swing API with its MVC paradigm and event oriented design
principles. It utilizes the models, events, and event listeners of
Swing and organizes the components as a hierarchy of containers with
layout managers.
<!-- </item:description index="1"> -->
```

Fig. 2. An HTML fragment after the insertion of meta-tags



```
- <rss version="2.0">
  - <channel>
    - <title>
        Enterprise Java Community: Your Enterprise Java Community
      </title>
      <link>http://dynamo.dynalias.org/tss.jsp</link>
    - <description>
        Enterprise Java Community is a developer community, containing up-to-date news,
        discussions, patterns, resources, and media
      </description>
    - <image>
        <title>TheServerSide.com</title>
        <link>http://www.theserverside.com</link>
      - <url>
          http://www.theserverside.com/skin/images/feed-logo.jpg
        </url>
      </image>
      <dc:language>en-us</dc:language>
    - <item>
        <title>wingS 2.0 web framework released</title>
      - <link>
          http://feeds.feedburner.com/techtarget/tsscom/home?m=315
        </link>
      - <description>
          The wingS project has just released version 2.0 of its framework with lots of
          major improvements. wingS is a component based web framework resembling
          the Java Swing API with its MVC paradigm and event oriented design principles.
          It utilizes the models, events, and event listeners of Swing and organizes the
          components as a hierarchy of containers with layout managers.
        </description>
        <pubDate>Thu, 08 Dec 2005 08:28:04 EST</pubDate>
      </item>
  </channel>
</rss>
```

Fig. 3. The fragment in RSS 2.0 format

meta-tags, which can be accomplished with a very little effort and/or change in currently available content management and publishing systems. It is beyond the scope of our application to be able to discover the appropriate patterns inside the HTML documents and *automatically* insert the meta-tags, which can be successfully done by our application only if the HTML document never changes in its internal structure.

Let us now see how a user interacts with the application. First of all, a user can verify the available RSS resources through his Web browser. She obtains a list of the available resources which can be formatted in one of the two currently supported formats, namely RSS 1.0 or 2.0. Following the link, the user gets the archive of the resource, chronologically ordered from the newest to the oldest. Our application allows also to aggregate RSS items and to query them. It is then possible to keep up-to-date by requesting a fixed number of the newest items. It is also possible to request the newest items containing a certain keyword in the title or in the description.

VI. Application experience

A. *The proof-of-concept: dynamo.dynalias.org*

We made a working prototype of our architecture, that we called Dynamo, available at http://dynamo.dynalias.org. By now Dynamo publishes the news feeds, in both RSS1 and RSS2 formats, taken from the Web portals www.serverside.com and www.java.net, each of which produces about 4-5 news (in plain HTML format) every day. In order to avoid any interaction with the portals we resorted to download the HTML pages containing the news, insert the meta-tags we defined and submit them to the entire procedure of extraction, storing and publishing.

B. *Scalability issues*

A typical problem in the design of an architecture like ours consists in the forecast of all possible critical elements that can raise as work loads become bigger and bigger. First of all it must be considered that an instance of Dynamo can be installed for each Web server. In those cases in which we have a very frequent production of news coming from different sources of information, it is possible to install a ”copy” of Dynamo for each source so to distribute and even balance the load. Another, even more severe, possible limitation to the performance of the proposed architecture is represented from the bandwidth required to forward the requests for updates, because in those cases of non regular updates a lot of requests would be useless thus resulting in wasting bandwidth. This is the reason of an improvement we are studying, that is a polling policy able to fit the frequency of the updates of the news from the Web servers: this policy, we called smart polling policy, adjusts the frequency of the requests for updates to the frequency with which Web portals generate new information.

Another factor that may affect the overall performances of DynamoNews is the host database management system, which is in our implementation is *eXist*, an Open Source native XML database whose performance seems not up to those of the DBMSs normally adopted to service Web portals. To avoid long response times even for simple queries, we have implemented a cache engine where the most frequently requested queries are stored.

We introduced two different polling policies, which can be chosen and plugged in our application independently from each other. The first is called ”flat” polling policy, as it does not depend from update frequency, while the second is called ”smart”, as it tries to fit the update frequency of each Web portal. It is possible to reconfigure at run-time the Poller component of the application (see further), in order to switch policy at runtime. It must however be considered that the smart polling converges asymptotically to the flat one.

With the flat polling policy, Web resources are queried for updates at regular time intervals which can be modified. It is the simplest strategy and it well applies to regularly updated information. The first improvement one can make over flat polling is to compute the frequency of the requests of updated Web documents as an estimate of the frequency. Then such estimate is compared to the *real* frequency with which Web documents are updated or newly generated. Both the estimate and the *real* times are used to compute a new estimate. That is:

$$\tau_{n+1} = \alpha\tau_n + (1-\alpha)t_n \qquad (1)$$

where τ_{n+1} is the estimate at the $(n+1)$-th iteration, τ_n the estimate at the n-th iteration, t_n the *real* frequency at the n-th iteration. The parameter α, whose value stands in the interval between 0 and 1, represents the relative weight of the previous estimate w.r.t. the *real* frequency. As τ_{n+1} takes into account the previous iterations, α represents the importance given to previous iterations.

Some considerations about the parameter α. Its value, comprised between 0 and 1, influences the velocity with which the frequency of polling equals the frequency with which Web portals publish new information. We found, on the other side, that its value does not influence the *convergence* of the frequency of polling to the frequency of publication, but only its velocity. Analogous results can be found in literature, even if in rather different situations. See, for example, the algorithm of processes scheduling known as *shortest job first* [11], as well as the weighed mean frequently used in iterative calculations typical of the Self-Consistent Integral Theories in Statistical Many-Body Thermodynamics. Please refer to the survey in

VII. Relationship with literature

The amount of information currently available in HTML format is really huge, but the main limitation in its fruition consists in its poor machine-readability, that is, in its lack of *structure.* Such problem can be solved, vis-a-vis the size of the Web and of individual Web portals, by making the extraction and annotation phase automated at least to some extent. To the best of our knowledge, the most advanced example of this approach is the LiXto [12] suite. LiXto supports the semi-automated creation of extraction programs, called filters, which, thanks to some clever logic-based representation of the HTML/XML structure [13], [14] of the document, is tolerant to some degree of elaboration of the source. We are planning to experiment with LiXto to make our extraction function capable of re-arranging the meta-tags annotation to adapt to changes in the HTML source.

VIII. Final considerations

We have described a Web application that generates and manages the RSS feeds extracted from HTML Web documents. The proposed architecture is intended to be applicable to arbitrary Web sites, provided that the Web administrator decides to start the service by adding the proposed meta-tags to the commented part of each page.

In order to collect the information relevant for the generation of a RSS feed we have defined a set of XML-like annotations which have to be inserted inside the HTML documents that contain the information we want to convert. The information is then extracted and organized into an XML format for storing. Typical actions which can be made include aggregation, query and conversion to RSS formats for syndication.

The most contemporary Web Content Management Systems (CMS) can handle news publishing and channeling by dynamic procedures which, upon user's request, retrieve data from the DBMS, the insertion of Dynamo meta-tags is accomplish just by some slight modification of those procedures (usually coded in PHP, JSP or ASP). Although content management systems of the last generation allow the publication of news in RSS format, Dynamo has the advantage of preparing and storing XML news and querying the database in a more *semantics-driven* way than with the relational databases which normally underlie CMSs.

We believe that there is room in the current landscape of the Web for this solution as it allows upgrading existing Web portals with minimal effort. As an instance, our recent work [15] describes how to bring a legacy system for the managing of community Web pages up to RSS news channeling. By hosting hundreds of discussion lists, accessed daily by thousands of users, the considered application is a good, and successful testbed for Dynamo.

References

[1] WC, “Rdf site summary (rss) 1.0.” [Online]. Available: http://web.resource.org/rss/1.0/spec

[2] “Rss 2.0 specification.” [Online]. Available: http://blogs.law.harvard.edu/tech/rss

[3] S. Bossa, *Towards the Semantic Web: a platform for dynamic generation, query and archival of RSS contents (In Italian). http://informatica.unime.it/*: Graduation Project in Computer Science, Univ. of Messina, 2005.

[4] R. T. Fielding, *Architectural Styles and the Design of Network-based Software Architectures*. Ph.D. Dissertation, 2000.

[5] R. L. Costello, “Building web services the rest way.” [Online]. Available: http://www.xfront.com/REST-Web-Services.html

[6] W3C, “Xsl transformations (xslt) version 1.0,” 11 1999.

[7] ——, “Xml schema part 0: Primer version 2.0,” 10 2004. [Online]. Available: http://www.w3.org/TR/xmlschema-0

[8] R. Bourret, “Xml and databases.” [Online]. Available: http://www.rpbourret.com/xml/XMLAndDatabases.htm

[9] J. M. Snell, “Resource-oriented vs. activity-oriented web services.” [Online]. Available: ftp://www6.software.ibm.com/software/developer/library/ws-restvsoap.pdf

[10] W3C, “Soap version 1.2 part 0: Primer,” 06 2003. [Online]. Available: http://www.w3.org/TR/2003/REC-soap12-part0-20030624/

[11] G. G. Abraham Silberschatz, Peter Galvin, *Operating System Concepts VI Edition*. John Wiley & Sons, 2002.

[12] G. Gottlob, R. Baumgartner, and S. Flesca, “Visual web information extraction with lixto,” *Proc. of VLDB Conference*, 2001.

[13] G. Gottlob and C. Koch, “Monadic datalog and the expressive power of languages for web information extraction.” *Journal of the ACM*, vol. 51, 2004.

[14] G. Gottlob and et Al., “The lixto data extraction project – back and forth between theory and practice.” *Proc. of PODS, Principles of Database Systems*, 2004.

[15] F. DeCindio, G. Fiumara, M. Marchi, A. Provetti, L. Ripamonti, and L. Sonnante, “Aggregating information and enforcing awareness across communities: the dynamo rss feeds creation engine,” in *Proc. of COM-INF06 Workshop*. Springer LNCS, 2006.

Building Agents with Agents and Patterns

L. Sabatucci [(1)], M. Cossentino [(2,3)], S. Gaglio [(1,2)]
(1) DINFO - Dipartimento di Ingegneria Informatica, Università degli Studi di Palermo - Viale delle Scienze, 90128 Palermo, Italy
(2) Istituto di Calcolo delle Reti ad Alte Prestazioni, Consiglio Nazionale delle Ricerche;
(3) SET - Université de Technologie Belfort-Montbéliard - 90010 Belfort cedex, France
sabatucci@csai.unipa.it; cossentino@pa.icar.cnr.it; gaglio@unipa.it

Abstract—**The use of design patterns proved successful in lowering the development time and number of errors when producing software with the object-oriented paradigm. Now the need for a reuse technique is occurring for the emergent agent paradigm, for which a great effort is currently spending in methodology definitions. In this work we present our experiences in the identification, description, production and use of agents patterns. A repository of patterns was enriched during these years so to request a classification criteria and a documentation template useful to help user during the selection.**

Index Terms—**Multiagent systems, patterns, reuse models and tools.**

I. Introduction

IN the last years, multi-agent systems (MAS) achieved a remarkable success and diffusion in employment for distributed and complex applications. In our research we focus on the design process of agent societies, activity that involves a set of implications such as capturing the ontology of the domain, representing social aspects, and intelligent behaviours. In the following, we will pursuit a specific goal: lowering the time and costs of developing a MAS application. We think that a fundamental contribution could come by the definition of reuse techniques and tools providing a strong support during the design phase. We identified in design patterns a good solution to this need. Significant motivations to the use of design patterns in a project are:

- **Patterns communicate knowledge**: they allow experts to document, reason and discuss systematically about solutions applied to specific problems. Patterns also help people to learn a new design paradigm or architectural style, and help new developers ignore traps and pitfalls that have been learned only by costly experiences [11].
- **Patterns increment quality of software**: design patterns are signs of quality because their use implies safe and elegant solutions that are validated by the experience rather than from testing [19].
- **Patterns improve the documentation process**: the pattern catalogue constitutes a documentation repository where the designer may explore possible solutions for his/her problem: each pattern provides a comprehensible way of documenting complex software architectures by expressing the structure and the collaboration of participants at a level higher than source code [20].
- **Patterns decrease development time**: design patterns are strategies helping people to find their way through complex situations by applying ready solution to solve difficult problems. Also they help in diagnosing, revising, and improving a group's work [11][14].
- **Patterns improve software maintenance**: a project obtained with patterns reuse is robust and simpler to modify with respect to traditional projects [19].

Our definition of pattern come from traditional object-oriented design patterns, revised for the agent paradigm. In particular we use an ontological approach, strongly influenced by the study of multi-agent system (MAS) meta-models.

In this paper we will present AgentFactory II, a tool for working with patterns for agents, integrating a user interface to select and apply patterns from a repository. AgentFactory II is based on the experience done with a previous release of the software [7] that was useful for exploring the possibility of designing a multi-agent system using design patterns as building blocks and successively to generate code from them. The major innovation of the tool is an expert system able to reason about the project and patterns, and a complex system to generate source code and documentation.

The paper is organized as following: in the section II we discuss the PASSI design process that is the base of our approach; in section III we introduce our agent patterns definition whereas in section IV we illustrate the architecture adopted to realize the tool; in section V we illustrate the *DocWeaver*, a specific agent of this society, that is responsible to generate the documentation in a specific agent-oriented style. Finally in section VI we report some conclusions.

II. The PASSI Design Process

In our work we will refer to the PASSI [4] methodology that represents the starting point and the natural context of our pattern definition and application. PASSI (Process for Agent Societies Specification and Implementation) drives the designer from the requirements analysis to the implementation phase for the construction of a multi-agent system. The design work is carried out through the construction of five models obtained by performing twelve sequential and iterative activities. Briefly, the phases and activities of PASSI are:

- **System Requirements**. It produces a description of the functionalities for the system-to-be, driving an initial decomposition of the problem according to the agent paradigm. The four activities are: (i) the Domain

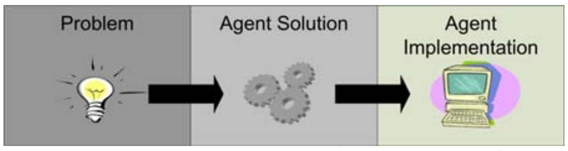


Fig. 1 – The three levels architecture for our pattern definition

Requirements Description, where the system is described in terms of the functionalities; (ii) the Agent Identification where agents are introduced for dealing with identified requirements; (iii) the Role Identification where agents' interactions are described by the introduction of roles; (iv) the Task Specification where the plan of each agent is draft.

- **Agent Society**. It is the phase where the agent paradigm is fully exploited. It is composed of four activities: i) in the Domain Ontology Description the system domain is represented in terms of concepts, predicates and actions; ii) the Communication Ontology Description focuses on the agents' communications, described in terms of referred ontological elements, content language and protocol; iii) in the Role Description the distinct roles played by agents are detailed within their dependencies.
- **Agent Implementation**. It is a model of the solution architecture in terms of required classes with their attributes and methods. It is composed of two main streams of activities (structure definition and behaviour description) both performed at the single-agent and multi-agent levels of abstraction.
- **Code**. It is a model of the solution at the code level. It is largely supported by patterns reuse and automatic code generation.
- **Deployment**. It is a model of the distribution of the parts of the system across hardware processing unit; it describes the allocation of agents in the units and any constraint on migration and mobility.
- **Testing.** It has been divided into two different activities: the Agent and the Society test. In the first one the behavior of each agent is verified with regards to the original requirements whereas during the Society Test, integration verification is carried out together with the validation of the overall results of the iteration.

III. AGENT PATTERNS

In order to work with agent design patterns we need a definition of what such a pattern is. We agree with the traditional object-oriented definition for design patterns, but we introduced some changes in order to adapt it for the agent paradigm.

We look at a pattern as "a problem which occurs over and over again in our environment, and then describes the core solution to that problem" [1]; the common use of design patterns is to describe best practices, good designs, and capture experience in such a way that it is possible for others to reuse them [11].

Our design patterns approach was conceived during the development of the PASSI process [4] with the goal of introducing a viable reuse technique for the development of MASs: our reuse technique uses some PASSI diagrams for describing the proposed solution. In this way the "solution" introduced is expressed in agent oriented terms, for instance agent, role, communication, goal and so on.

Table 1 – Description for the GenericAgent pattern

Name: GenericAgent
Classification: internal architecture/single-agent
Intent: this pattern may be used as the root before applying all single-agent patterns because it gives to an agent the ability of registering/deregistering to the platform services (AMS and DF).
Motivation: this pattern is useful for agents who want to discover if the system offers a specific service and what agents can provide it. The GenericAgent pattern adds the ability of registration to the platform (white/yellow pages) so that the agent is accessible for conversations.
Preconditions: none.
Postconditions: the agent is able of registering and de-registering to AMS e DF.
Solution (Structure, Participants and Collaboration): the target agent is enriched with an attribute for listing the description of all its services offered to the community. A *registerDF()* and *registerAMS()* methods with their correspondent *deregisterDF()* and *deregisterAMS()* are introduced to agent class.
Related Patterns: this pattern may be the predecessor for all single-agent patterns. The LogAgent is a variant of this pattern which may be used specifically for debugging/testing aims.

Jackson in an analysis of software design phases [15] distinguishes between the problem and the solution context: the problem and its solution are separated entities located in two different conceptual positions. The solution stays *in the computer and in its software* (machine domain) whereas the problem is *in the world outside from it* (application domain). Our approach to the definition of agent patterns spreads across both of the application and machine domains. However we need to specialize the Jackson's domains to cope with the agent concept. When using agents as a design paradigm the solution is generally quite abstract with respect to its expression in terms of object oriented concepts. We split the machine domain in two sub-domains, introducing the "agency domain" between the problem and the implementation domains (see Fig. 1). Our pattern architecture is based on these three levels:

Pattern problem. A fundamental part of a pattern is the textual description of the problem for which it may be useful. It is composed by: (i) *motivation*, an explanation of how (and why) the pattern works, and why it is good, putting into evidence steps and rules required to resolve the problem; (ii) the application context describes the conditions under which the problem and the solution seem to recur, and for which the

Table 2 - Rules for the GenericAgent pattern

```
(deffunction generic_agent (?name)
   (if (generic_agent_precond ?name) then
      (add_new agent ?name)
      (add_new agent_action “register_DF” ?name)
      (add_new agent_action “unregister_DF” ?name)
      (add_new agent_action “register_AMS” ?name)
      (add_new agent_action “unregister_AMS” ?name)
   ))

   (deffunction generic_agent_precond (?name)
      (if (exist (agent ?name)) then
         (return FALSE)
      else
         (return TRUE)
   ))
)
```

solution is desirable; (iii) *related patterns* element describes other patterns that could solve a similar problem. As an instance of pattern we report the *GenericAgent* described in details in Table 1.

Pattern solution. It represents the solution (introduced when adopting the pattern) in terms of agent-oriented elements. The solution description illustrates the static structure and the dynamic behaviour introduced by the pattern in terms of resources, participants and collaborations. The formal description is a set of rules expressed using a logical language based on Jess. These rules are classified in three groups: i) the preconditions have to be verified before to introduce the pattern, ii) the postconditions are rules to verify after the pattern application (they may condition future patterns application), and iii) the solution rules that are a logical description of the elements constituting the solution and their behaviour/interactions. Our patterns for agents are explicitly defined to be used in conjunction with the PASSI methodology [4]; as a consequence the solution is described using some diagrams from the PASSI phases depicting agents' internal structure and social behaviour. Roles, tasks, communications, and interaction protocols are examples of the involved elements. An instance of rules for the pattern solution for the previously introduced *GenericAgent* is shown in Table 2; in the subsection IV.B we will describe how these rules influence the design when the pattern is introduced in the project.

Pattern implementation. This represents the lower level of the solution containing the effective implementation in object oriented terms. It uses diagrams of PASSI depicting the static structure of the involved agents in terms of classes, attributes and methods using conventional UML class diagrams and dynamic behaviour of one or more agents involved in interactions using activity or state-chart diagrams.

The main feature of our tool is to automatically generate the solution at this implementation level. This feature will be discussed in the subsection IV.C.

IV. THE AGENT FACTORY TOOL

The AgentFactory II tool was designed and developed after some experiences done developing and using the previous version of the tool [5][8]. The strategic choice distinguishing this new version of the tool from the previous one is that we are developing it as a multi-agent system.

The system as shown in Fig. 2 is basically composed by four agent organizations [11] (or groups of agents responsible of a functional area): i) the pattern architect, ii) the agent model, iii) the aspect weavers and iv) the object model. Each organization will be discussed in details in the following subsections. The *UserAgent*, external to all these organizations, is responsible to interact with the designer, using a GUI (a screenshot is reported in Fig. 3); this agent has the goal of adapting its GUI to the agents present in the system (that are not a-priori known); in order to deal with an ontology that is not a-priori known we used an high level ontology an reflection techniques [21][2]. In Fig. 3 we show an instance of the *UserAgent* GUI: the tree on the left panel reports the model hierarchy of the project; in the right panel it is possible to manually edit data for the element selected in the tree (often elements are introduced using patterns); specifically, in the example, the *ParticipantRole* role is selected and the right panel shows text-fields for this element: the role name, the author and the documentation, the agent who plays the role, the tasks involved in the role and finally some custom attributes.

A. The Agent Model Organization

This organization is responsible to manage the “agent solution” level of our architecture (reported in Fig. 2). This organization is designed to front a hard problem: maintaining the meta-model of our patterns independent from the specific methodology employed to design a system. This is a hard goal because all the agent-oriented methodologies use specific meta-models, involving different concepts or assigning them different meanings.

We structured the “Agent Model” as a holonic organization [12] (shown in Fig. 4) based on three basic roles (that are played by the agents of the organization): i) the MMDF is the head of the hierarchy, ii) the Fragment Agents stay at the intermediate level, whereas iii) the Model Agents are the bodies of this holonic structure.

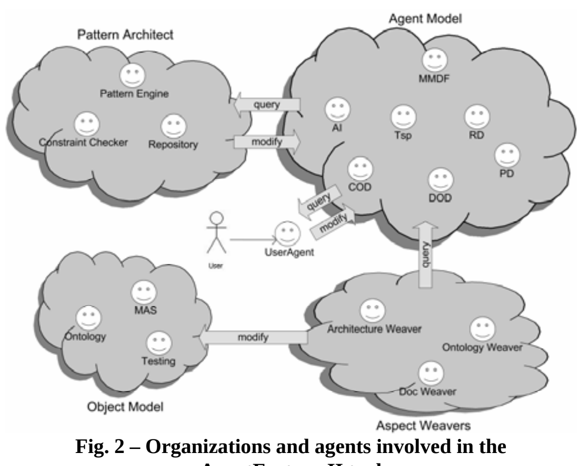
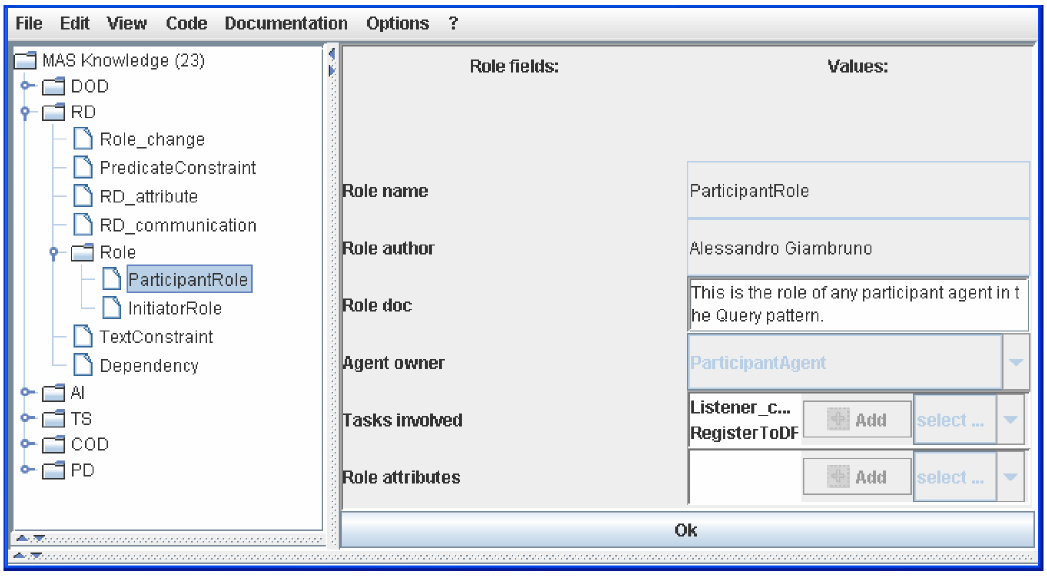


Fig. 2 – Organizations and agents involved in the AgentFactory II tool



Role fields:	Values:
Role name	ParticipantRole
Role author	Alessandro Giambruno
Role doc	This is the role of any participant agent in the Query pattern.
Agent owner	ParticipantAgent
Tasks involved	Listener_c... RegisterToDF
Role attributes	

Fig. 3 - A screenshot of the AgentFactory UserAgent

The most important role of the organization is played by the MMDF (MetaModel Directory Facilitator) agent, that is inspired to the FIPA [8] Directory Facilitator (DF); in the abstract architecture defined by FIPA, the DF is the agent responsible to maintain the yellow pages for all the services in the system by communicating with the DF all the agents may register their own services or discovery services offered by other agents. The MMDF agent has a similar function but focused on building the meta-model used during design: at the beginning the meta-model is empty; when the model agents are executed they register one or more meta-model elements: therefore the MMDF is populated at run-time (according to a specific methodology).

Fragment agents represent "pieces of a methodology" and are responsible to group model agents coming from the same methodology in a model holon; this was done for two motivations: i) fragment agents coordinate the work among their model agents (internal collaboration); ii) fragment agents enable the collaboration of elements coming from different methodologies (external collaborations). For illustrating this concept, in Fig. 4 we show a possible configuration for the "Agent Model" organization. We have two fragments coming from two agent oriented methodologies: PASSI [4] and Tropos [4]. Each of these fragments is responsible for different elements of the meta-model (*requirement*, *role* and *agent* for PASSI, *goal*, *resource* and *agent* for Tropos); intersections among model agents may be treated in two different ways: a concept may be shared among different fragments (as the *agent* in Fig. 4) or may be exclusive of a methodology.

B. *The Pattern Architect Organization*

This is the organization responsible for managing the pattern repository and introducing selected patterns into the system. Our pattern implementation is realized using a first order language; we have chosen to extend the Jess language [16] that is a lisp-like language, adding the ability to access to the services offered by the Agent Model (for instance to query for a specific element, or to introduce a new element). In Table 2 there is an example of a pattern: the *GenericAgent*; that is used for giving to an agent the ability of registering/deregistering in/from the platform services (white/yellow pages). This pattern is useful for agents who want to discover if the system offers a specific service and what agents can provide it. The pattern is done by a rule, *generic_agent*, that is activated using a parameter (the name of the new agent). This simple set of rules verifies (precondition) if an agent with the same name exists in the project, an then (pattern solution) adds the agent with some abilities (*register_DF*, *unregister_DF*, *register_AMS*, *unregister_AMS*). In this example there are no postconditions.

C. *The Aspect Weavers Organization*

A significant characteristic of AgentFactory (already present in the early version of the tool) is the automatic code generation for different platforms (until now we supported only Jade [2] and FIPA-OS [10], but it was conceived for being extended with other agent-platforms that are compliant with the abstract FIPA architecture [8]). The previous version of the tool had a code generation engine based on a sequence

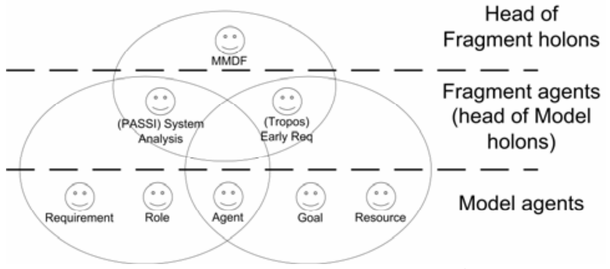
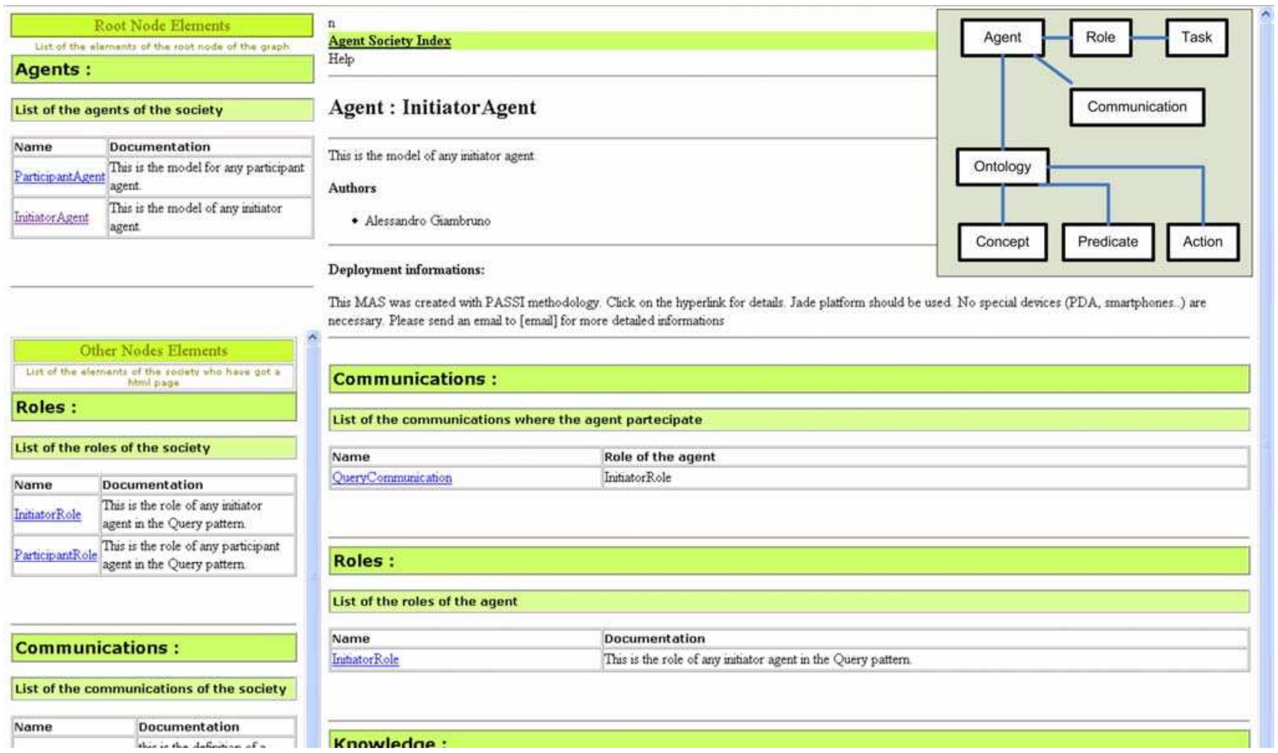


Fig. 4 – Agents and roles in the holonic structure for the "Agent Model" organization



Root Node Elements

List of the elements of the root node of the graph

Agents :

List of the agents of the society

Name	Documentation
ParticipantAgent	This is the model for any participant agent.
InitiatorAgent	This is the model of any initiator agent.

Other Nodes Elements

List of the elements of the society who have got a html page

Roles :

List of the roles of the society

Name	Documentation
InitiatorRole	This is the role of any initiator agent in the Query pattern.
ParticipantRole	This is the role of any participant agent in the Query pattern.

Communications :

List of the communications of the society

Name	Documentation
	this is the definition of a

n
Agent Society Index
Help

Agent : InitiatorAgent

This is the model of any initiator agent.

Authors

- Alessandro Giambruno

Deployment informations:

This MAS was created with PASSI methodology. Click on the hyperlink for details. Jade platform should be used. No special devices (PDA, smartphones..) are necessary. Please send an email to [email] for more detailed informations

Communications :

List of the communications where the agent partecipate

Name	Role of the agent
QueryCommunication	InitiatorRole

Roles :

List of the roles of the agent

Name	Documentation
InitiatorRole	This is the role of any initiator agent in the Query pattern.

Knowledge :

Fig. 5 - The portion of the PASSI MAS meta-model used to generate the documentation with the DocAgent; on the right a screenshot of the hypertextual documentation generated for the case study

of transformations according to the MDA architecture [17]. In this new version we are realizing a more complex transformation engine, that is inspired to Aspect Oriented Programming (AOP) [17] in order to reduce the gap between the agent solution (introduced using patterns from the repository and refined by the designer) and the object-oriented solution (that is typically an object oriented system). We referred to collaborative team-work as a metaphor for where different human-roles (that are expert in their own sector) individually work in a specific competence area, giving their personal contribution to the final solution. In our context agents are the experts and each area of competence is an aspect of the agent-oriented solution to take in consideration for code production. Agents have to collaborate in order to converge all their single contribution in the same final object-oriented code. In the AOP terminology the engine realizing this convergence is called 'aspect weaver'; this is the motivation for the name chosen for this organization: an aspect weaver agent is the 'expert' of a specific area of the project; it is responsible to a specific aspect of the project and it is able to generate an output in terms of object-oriented solution. The entire organization is organized to weave all the contributions coming from different agents and to meet them in an unique solution.

We actually realized only three weaver agents: i) an *ArchitectureWeaver* (responsible of the agent skeleton and communications), ii) an *OntologyWeaver* (responsible to add ontology to the messages exchanged by agents) and iii) a *DocWeaver* (that creates the documentation; it will be discussed in details in section V). The *ArchitectureWeaver* fundamentally carries out the code generation functionality of the previous version of AgentFactory, generating the base architecture of the agents within their abilities/tasks. The *OntologyWeaver* adds the management of the ontology: concepts, predicates and actions that are used in the agent knowledge and communications.

D. The Object Model Organization

This organization is conceived for realizing the agent implementation level of our architecture (see Fig. 2); it is relative to the object oriented solution. Agents of this organization are responsible to treat elements of the object oriented paradigm (such as classes, methods, attributes and so on). The organization is composed by three agents: i) the *MAS* agent, ii) the *Ontology* agent and iii) the *Testing* agent. The *MAS* agent is responsible to handle data of a whole multi-agent system taking in consideration both the static structure of the agent and the behaviour of the multi-agent system. The organization is able to export the source code for Jade and FIPA-OS agent platforms. The *Ontology* agent is responsible to generate classes for the system ontology: these are serializable classes that are used in the agents' knowledge and communications. The *Test* agent (still under development) will be responsible to generate stub and driver agents for simulating the communications and collaborations among system agents (integration testing).

V. A WEAVER AGENT: DOCWEAVER

In the past, during the development of multi-agent systems we suffered the lack of a specific technique for documenting our source code; we used Javadoc for generating the API documentation (from comments in source code), but we noted it is difficult to navigate because it implies a shift in the paradigm (from agent-oriented to object-oriented and vice-versa); whereas the solution is expressed in agent oriented terms, the documentation is expressed in object oriented terms: the mapping is not direct and easy. Therefore we demanded a way for documenting our solution using directly an agent oriented style.

From these considerations we deducted the requirements for the AgentDoc, an agent oriented style for documenting a multi-agent system; the terms included in this documentation are not fixed, but are depending from the specific methodology used and therefore from the specific MAS meta-model adopted. AgentFactory II is naturally inclined to use different meta-models, so we create a *DocWeaver* agent responsible to generate the AgentDoc for each designed MAS. In order to generate this documentation the AgentDoc uses the meta-model stored in the MMDF. This is not enough because the agent requires information about how an element of the MAS meta-model influences the documentation content. In order to solve this problem the *DocWeaver* uses a (manually built) configuration graph that specifies what elements (graph nodes) have to be included in the documentation (for each instance of the included elements an HTML page is generated); whereas the relationships among the elements (graph arcs) generates links among pages: the result is a navigable hypertextual documentation.

In the grey box in Fig. 5 we report the graph used for generating the documentation for a PASSI project. It is a simplified version of the PASSI meta-model composed by Agent, Role, Task, Communication and Ontology. Fig. 5 shows an example of the generated documentation concerning an agent of the system; the page presents a left frame with a list of the system elements (agents, roles…), and a right frame with details of the selected item; for instance we focus on an agent and its details are: roles, communications and ontology (that are the nodes with distance one from the agent node).

VI. Conclusions And Future Work

Our conviction is that pattern reuse is a very challenging and interesting issue in multi-agent systems as it has been in object-oriented ones. However we are aware that the problems arising from this subject are quite delicate and risky. Nonetheless, we believe, thanks to the experiences we made in application fields such as informative systems and robotics, that it is possible to obtain great results with a correct approach.

In order to support the design of multi-agent system we developed a complex multi-agent system for building agents with a pattern support. This tool is also able to generate the documentation and the source code for the project. Actually the code generated is just a bit richer that the code generated in the previous version, however we are working on a more complex organization with a greater number of weaver agents involving other aspects as role, task, plan and so on; in this context we require a more precise coordination mechanism among the weavers. Another improvement under development is the *Testing* agent, that would be employed for integration testing on multi-agent system.

References

[1] Alexander C. 1979. The Timeless Way of Building. Oxford University Press

[2] Arnold, K. and Gosling, J. 1998. The Java Programming Language (2nd Ed.). ACM Press/Addison-Wesley Publishing Co.

[3] Bellifemine F., Poggi A. and Rimassa G. 2001. Developing Multi-agent Systems with JADE. In proceedings of The 7th international Workshop on intelligent Agents. Agent theories Architectures and Languages (July 07 - 09, 2000), LNCS 1986, Springer-Verlag, London, pp. 89-103.

[4] Bresciani P., Giorgini P., Giunchiglia F., Mylopoulos J., and Perini A. 2004. TROPOS: An Agent-Oriented Software Development Methodology, Journal of Autonomous Agents and Multi-Agent Systems, Kluwer Academic Publishers 8(3), pp. 203-236

[5] Chella A, Cossentino M and Sabatucci L. 2003. Designing JADE systems with the support of CASE tools and patterns. Exp Journal, Sept; 3(3):86–95.

[6] Cossentino M. 2005. From Requirements to Code with the PASSI Methodology, In Agent-Oriented Methodologies, edited by B. Henderson-Sellers and P. Giorgini, Idea Group Inc., Hershey, PA, USA

[7] Cossentino M., Sabatucci L. and Chella A. 2003. A Possible Approach to the Development of Robotic Multi-Agent Systems. IEEE/WIC Conf. on Intelligent Agent Technology (IAT'03). Halifax (Canada), October, 13-17, pp 539- 44.

[8] Cossentino M., Sabatucci L. and Chella A. 2003. A Possible Approach to the Development of Robotic Multi-Agent Systems - IEEE/WIC Conf. on Intelligent Agent Technology (IAT'03). October, 13-17, 2003. Halifax (Canada).

[9] FIPA Abstract Architecture – [Available on Internet] http://www.fipa.org/repository/architecturespecs.html

[10] FIPA-OS Website - [Available on Internet], http://fipaos.sourceforge.net

[11] Ferber J., Gutknecht O. 1998. A Meta-Model for the Analysis and Design of Organizations in Multi-Agent Systems. In Third International Conference on Multi Agent Systems (ICMAS'98); p 128.

[12] Fischer K., Schillo M. and Siekmann J. 2003. Holonic Multiagent Systems: A Foundation for the Organisation of Multiagent Systems, Lecture Notes in Computer Science, Volume 2744, Jan 2003, Pages 71 – 80

[13] Gamma E., Helm R., Johnson R., and Vlissides J. 1994. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.

[14] Greenfield J. and Short K. 2003. Software factories: assembling applications with patterns, models, frameworks and tools. In Companion of the 18th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (Anaheim, CA, USA, October 26 - 30, 2003). OOPSLA '03. ACM Press, New York, NY, pp. 16-27

[15] Jackson M. 2001. Problem Frames: Analysing and Structuring Software Development Problems, Addison-Wesley

[16] Jess Rule Engine – available at: http://herzberg.ca.sandia.gov/jess/

[17] Kiczales, G. 1996. Aspect-oriented programming. ACM Comput. Surv. 28, 4es (Dec. 1996)

[18] OMG Model Driven Architecture - [Available on Internet] http://www.omg.org/mda/

[19] Prechelt L., Unger B., Philippsen M. and Tichy W. 2002. Two Controlled Experiments Assessing the Usefulness of Design Pattern Documentation in Program Maintenance. *IEEE Trans. Softw. Eng.* 28(6), pp. 595-606.

[20] Schmidt D. and Stephenson P. 1995. Experience Using Design Patterns to Evolve Communication Software Across Diverse OS Platforms, In proceedings of the *9th European Conference on Object-Oriented Programming*, LNCS 952, pp. 399 – 423

[21] Sun Microsystems. Java reflection. 2002. Available on internet at http://java.sun.com/j2se/1.3/docs/guide/reflection/index.html

A Repository of Fragments for Agent Systems Design

Valeria Seidita[1], Massimo Cossentino[2,3] and Salvatore Gaglio[1,2]

[1]Dipartimento di Ingegneria Informatica - University of Palermo
Viale delle Scienze 90128 Palermo, Italy
[2]Istituto di Calcolo e Reti ad Alte Prestazioni, Consiglio Nazionale delle Ricerche
Viale delle Scienze, 90128 Palermo, Italy
[3]SET - Université de Technologie, Belfort-Montbliard
90010 Belfort cedex, France
seidita@csai.unipa.it, cossentino@pa.icar.cnr.it, gaglio@unipa.it

Abstract

The creation of a new design process for a specific situation using the method engineering approach is based on the composition of a set of reusable method fragments. The request for these reusable method fragments leads to the need for a repository containing standardized fragments that can be easily selected and assembled in new design processes. In this work we present a definition of method fragment coming from the work of the FIPA Technical Committee Methodology and a repository where fragments are classified according to the specific process component (activity, process role, and work product) they underpin and on the specific MAS Metamodel element(s) they work on.

1 Introduction

Today a relevant number of design processes for developing multi agent systems can be found in literature; each of them is well suited for a specific purpose or for a specific agent architecture (BDI, reactive, state-based,...); one unique (and eventually standardized) design process fitting all possible situation does not exist; in the agent-based development context, we are now facing the same problem some researchers faced a few years ago when the definition of a new discipline was given, the Method Engineering.

Method Engineering aims at solving the previously said problem focusing on the creation of new techniques and tools allowing the construction of a specific design process (in literature referred as a situational method) [18]. Many researchers applied this paradigm and shared similar approaches: constructing and adapting new design processes by assembling (reusing) *method fragment* from a repository (called *method base*) built by splitting up some existing design processes [16][22][21][3][17].

Method engineer is the key stakeholder during a process construction activity; he develops two main phases, the first one regards extracting, defining, standardizing and storing in the repository the method fragments coming from existing processes while the second one consists in composing the new process through the selection and the assembly of the right fragments.

Our activity in this field started a few years ago within the FIPA Technical Committee (TC) Methodology from where the basis of this work arose. More specifically we acknowledge a great dependence of our method fragment definition with the one proposed in the FIPA context and also the design processes we studied are among the most important in that context (Adelfe, Gaia, PASSI, Tropos).

In this paper we introduce the repository we used for storing the fragments extracted from the above cited design processes. It is essentially a database where method fragments are stored in form of text documents and can be accessed using a categorization based on the process metamodel elements that we consider central in agent-systems design (process role, phase/activity, work product and MAS Metamodel element). It is interesting to note that, in our opinion, this latter element (the MAS Metamodel element, MMM element hereafter) is one of the keys of agent-system design today. The lack of a standardized or at least widely accepted MAS Metamodel brings to several different interpretations for it. Method engineers cannot neglect this aspect and we think that one of the first activities while building the new process is defining the MMM elements that he will instantiate and their relationships.

One of the most difficult activities in constructing our repository was its conceiving in such a way that fragments could be easily retrieved. We think that our solution to this problem, coherently with the choice of basing categorization on the four cited elements of the process metamodel is interesting: we built a taxonomy within each of the four basic categories (process role, phase/activity, work product, MMM element). In this way, a method engineer who aims at retrieving a fragment that produces a structural diagram (a kind of work product in our taxonomy) and involves a specific process role (like the Domain Analyst that is another item of our taxonomies), can easily find a list of all the fragments in the repository satisfying these criteria. A similarly interesting search could be related to the need for designing some kind of MMM element (suppose ontological concepts) because the method engineer wants to introduce a fragment about that in his/her new process.

The paper is organized as follows: in the next section we introduce our method fragment definition, in section three we describe the structure of our repository and in section four we provide an overview on the content of our method base; finally in section five, some conclusions are drawn.

2 Method Fragment

FIPA TC Methodology approach shares a similar meaning of method fragment with Harmsen and Brinkkemper [3][16][14]. A method fragment is a portion of a design process composed of two main parts, the process and the product. In our specific approach (also grounded on the process model proposed by an OMG specification, SPEM [20]) main process elements are Activity, Role and Artefact; more explicitly a development process is composed of Activities performed by one (or more) Role(s) responsible for producing artefacts, Activities produce or consume Artefacts as inputs or outputs. From now on, in order to be compliant with SPEM notation, we will refer to WorkProduct and ProcessRole in place of, respectively, Artefact and Role.

According to our approach a method fragment is composed as follows :

1. A portion of process (what is to be done, in what order), defined with a SPEM diagram.
2. One or more deliverables (WorkProducts like (A)UML/UML diagrams, text documents including code and so on). The result of the work could also be some kind of product/artefact that is not be delivered to anyone outside the development process. It also includes a reference to a recommended notation/language/ structure to be used.
3. Some preconditions (they are a kind of constraint because it is not possible to start the portion of process specified in the fragment without the required input data or without verifying the required guard condition).
4. A list of concepts (related to the MAS Metamodel) to be defined (designed) or refined during the specified process fragment.
5. Guideline(s) that illustrates how to apply the fragment and best practices related to that.
6. A glossary of terms used in the fragment (in order to avoid misunderstandings if the fragment is reused in a context that is different from the original one).
7. Composition guidelines are a description of the context/ problem that is behind the portion of methodology from which the specific fragment is extracted; it can be used to facilitate fragment reuse in the proper context.
8. Aspects of fragment are a textual description of specific issues like for instance: implementation platform for which the fragment is more suitable, application area, etc.
9. Dependency relationships that can be used to identify fragments strongly related to the considered one.

It can be represented by the metamodel shown in Fig. 1 where the presented elements are logically divided in three areas, the first one concerns the fundamental process features (Activity, ProcessRole, WorkProduct and MMMElement- drawn in grey in the figure), the second one concerns the reuse features of the fragment and the third one its representation in the repository.
As regards the first area, the main element is the Fragment that is part of a Development Process based on a well defined LifeCycle (for example waterfall[13], iterative/incremental[2]); Lifecycle, in the area concerning the reuse of the fragment, allows positioning the fragment in the proper place in the development cycle.

A fragment is composed of elements such as Activity, ProcessRole and WorkProduct useful for the description of the portion of process related to the specific fragment: an Activity describes a portion of work, performed by a ProcessRole [20] during which some data are used as input or produced as output, besides an Activity is an element of a Phase and it is composed of Steps, which are the smaller parts of work to be performed. At last an Activity can produce one or more WorkProducts, whatever consumed, produced, modified or refined during the portion of work considered in the fragment; a WorkProduct can be a diagram (for example an UML diagram) or a text document, and

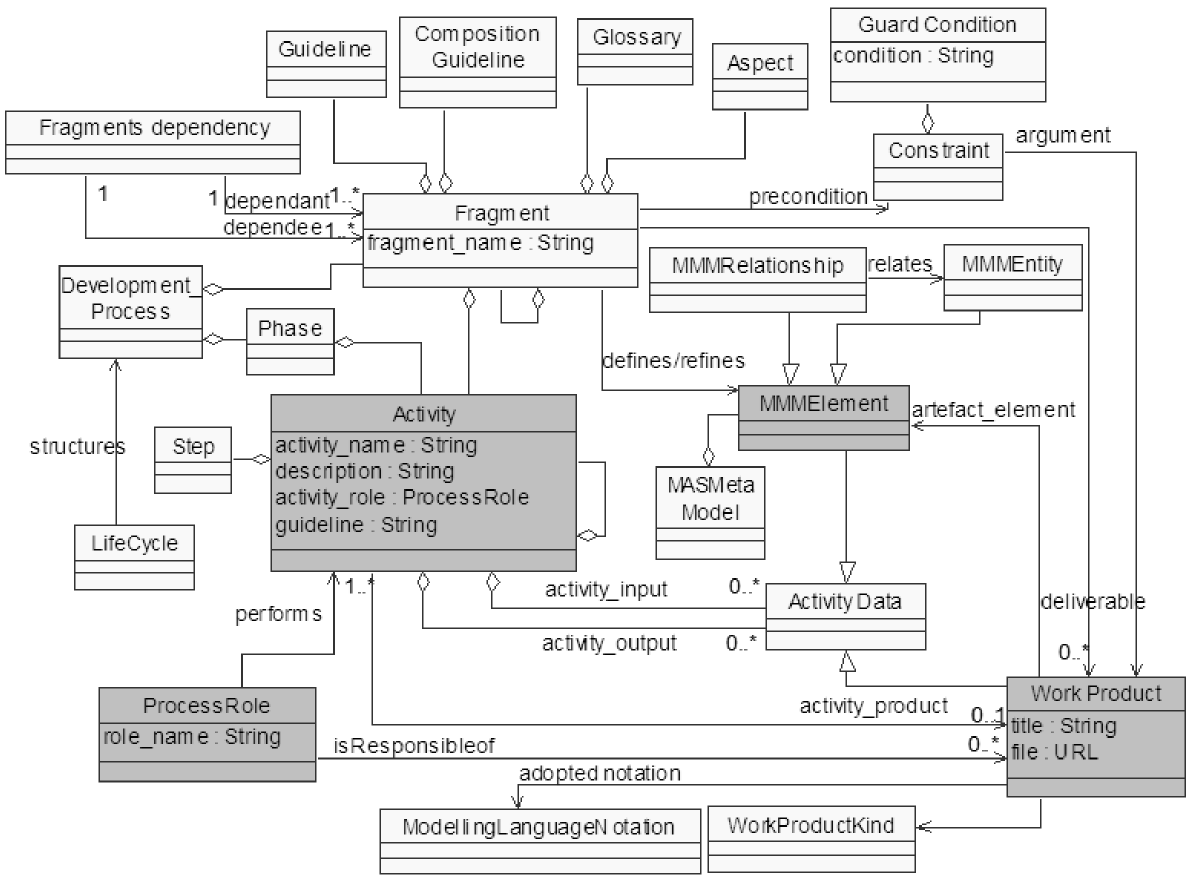


Figure 1. The method fragment metamodel

refers to a Modelling Language Notation. A WorkProduct is classified according to a WorkProductKind that describes the WorkProduct category, for instance text document, model, code library etc., it is built by a ProcessRole, which is responsible for its production. MMMElement (the Multi agent system Meta-Model Element) is a very important element of the fragment metamodel we adopted, it is the generic component of a MAS Metamodel (Multi Agent System metamodel) which constitutes the main building block of an agent design methodology. A MAS Metamodel is a structural/ontological representation of the elements (for instance agent, role, behaviour, ontology, ...) and relationships that will be instantiated in an actual system; instances of MAS Metamodel elements are described in WorkProducts and are also considered as input/output of activities; it is worth to note that in our approach the ultimate aim of the work performed in a fragment is to define/refine/relate MMMElements.
Two elements of the definition, belonging to this area, are not explicitly presented in the metamodel, but included in other elements, they are: portion of process, it is represented by the activities that are parts of the whole process, and list of concepts, which is composed of the set of MMMelements that are designed during the portion of process the fragment represents.
The second area (in Fig. 1) shows all the elements allowing the fragment reuse and assembly, they are the Glossary of terms used in the fragment, the Aspect, in the form of a textual description useful to identify the field of fragment application (several fragments are rather specific for some kind of agent architecture like the BDI one or an implementation platform like JADE and are not suitable for different contexts without a significant maintenance), and the Composition Guideline (while Aspects deal with the target system features, Composition Guidelines concern the development process and describe how and in which context the fragment can be profitably reused).
The third area describes the fragment as it is stored in the repository, Fragment Dependency is the only element it contains, this is a list of fragments that have a dependant or dependee relationship with the the specific reused fragment.

This definition of fragment is the basis for the extraction of method fragments from existing methodologies; in our opinion the extraction process aims at reifying the concepts of activity, role, work product and MAS Metamodel element. A fragment is physically stored in the repository in the form of a text document [7]; in these documents, SPEM is used as a process modeling language, in particular SPEM notation is employed to represent the main fragment elements (Activity, ProcessRole and WorkProduct) and some diagrams (for instance SPEM Activity Diagrams) are used to depict the work flow performed in the fragment and the relationships among these elements.

3 Fragment Repository

As we said in section 1 method engineering is a discipline which aim is to design, construct and adapt methods for information systems development [3][22][18]; one of the most common applications of the method engineering paradigm is an approach based on reuse, where the constituent parts of a development process (method fragments) are stored in a repository (method base) from which they could opportunely be selected and retrieved in order to be successively assembled in the new required process (this operation sometimes requires an adjustment of the fragment in order to properly place it in the new process).

In literature we found two repositories of fragments, the first one [15] is a method base associated to the Decamerone CAME tool and the second one is the OPF repository [11]. These repositories share the same aim: to facilitate the selection and retrieval of method fragments. In constructing our repository we share the same aim of the two previously cited examples but our approach is quite different.
The method base in Decamerone is based on the assumption that a method engineer creates a repository of fragments coming, only, from processes suitable for solving a particular problem; this brings about a first selection of design processes. The extraction of method fragments, that once stored in the repository may be selected and assembled in the new design process, regards only a limited number of fragments. Instead we aim at collecting a relevant number of the existing agent design processes and at storing all the fragments we can extract from them, in so doing, during the creation of a specific design process, we have a larger repository to be used and we can select and eventually adopt a fragment coming from a process that could not fit, in its wholeness, the problem we want to solve; we think this choice constitutes a richness for the selection activity. As regard OPF, which is the most important element of the OPEN approach [11], it comprises a metamodel representing all process elements which instantiation generates a method fragment; OPF contains a very large number of method fragments accessible from a website and it provides methods fragments at a very low level of granularity.
In our previous works [8][9][10][12] we extracted a lot of method fragments coming from different agent design processes created by different research groups and suited to deal with very different multi-agent system design philosophies, they are: Adelfe, Gaia, PASSI and Tropos [1][24][6][4].
All of these processes present substantial differences in the terms they adopted and in their meanings for specifying the design process elements; for instance in Adelfe process the process role called Requirement Analyst in some activities performs the same work performed by the System Analyst in the PASSI process; besides each process underpins a specific MAS Metamodel which elements have a meaning that can be different from the corresponding one in another process even if sometimes they share the same name. Therefore the fragments, once stored in the repository, lose importance and usefulness if we do not dispose a method for their easy retrieval. We thought that the rationale for storing the fragments in the repository (and then make clever query) was to consider the work performed by method engineer when trying to assemble new methodologies, he selects only the fragments he can really use; with this we mean that, for instance, if there is not any ontology designer among the workers of an organization, it is useless to include in the process under construction an activity to be performed by such a stakeholder (this would bring to a process that for its application would need skills that are unavailable). We think to facilitate the discrimination of the right fragments classifying them in categories based on the main process elements (2): Activity, ProcessRole, WorkProduct and MMMElement. However while categorizing the fragments we met a great problem: from the studied processes we collected about sixteen different process roles, seventeen phases (each of them is composed of several activities), a lot of work products and of MMM elements. Therefore we thought useful to categorize all the available method fragments according to a taxonomy unifying (and mapping) different elements (from different approaches) under a unique definition. We firstly identified the set of common activities, a design process is usually composed of, referring to the main phases of a software engineering design process [23][13], then the principal process roles performing these phases and finally a set of work product kinds; Fig. 2 shows the clustering rationale for phase and process role, it is just an illustrative representation of our taxonomy that does not exclude some possibility of intersection among different areas; for the work product kinds taxonomy we adopted the structure shown in Fig. 3. In the following subsections we will better illustrate the taxonomies.

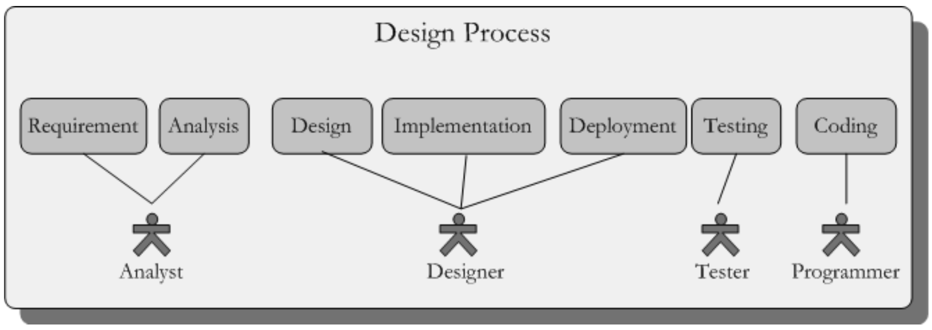


Figure 2. Categories for phases and process roles in the design process

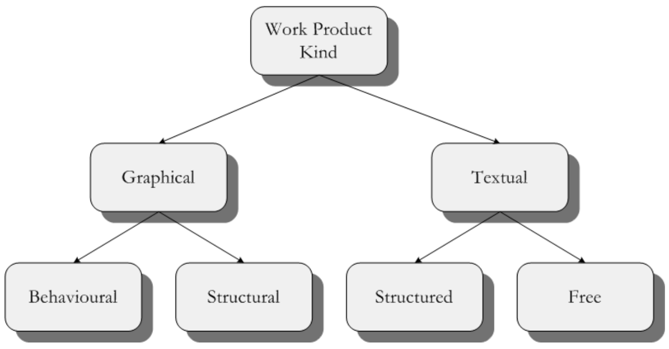


Figure 3. Categories for work product kind

3.1 Phase

Any kind of design process for information system production, and in our case for multi agent system production, can be decomposed in a set of activities (or phases) organized in sequential steps depending on the specific chosen process model; regardless of their organization some kind of activities have to be performed to develop any system. We examined all the phases our the 4 studied processes present (in terms of the work they carry on and their aim) and, referring to the main phases of a software engineering process, we clustered them in the following phases:
Requirements, it consists in the requirements elicitation phase during which a functional model is given to provide the purpose of the system and the interactions between the system and the environment.
Analysis, it consists in all the activities aiming at understanding the system and its structure (without reference to any implementation detail), identifying and defining the main entities of a MAS (such as role, communication, etc.).
Design, the aim of this phase is to define the agent architecture, describing agents' behaviours and to investigate how a society of agents cooperate to realise the system- level goals, and what is required of each individual agent in order to do this; all the aspects of the agent society are faced.
Implementation, gives a view on the system architecture, methods and classes are used to describe the agent's structure and behaviour.
Testing, it is composed of a unit test, the verify of the single agent's behaviour with regards to the original requirements of the system and a society test, the validation of efficient cooperation between agents.
Deployment, this phase defines and describes how agents are deployed and which constraints are present for their migration and mobility.
Coding, the phase of writing the code eventually with the aid of reusable code and source code.
Some of these phases are fundamental in a classic software development process while some others are specific for the agent oriented context, for instance design phase deals with the concept of agent and explores the social aspect of a multi agent system while a classic design phase concerns the way the different system components provide system functionalities.

3.2 Process Role

Starting from the phases identified in the previous subsection and from the examination of all the existing stakeholders in the referring agent design processes, we clustered, under the same element, a set of process roles performing similar activities. First of all we associated for each phase a process role, in Fig. 2 we can see that a general role called Analyst performs the requirement and analysis phases, the Designer performs design, implementation and deployment, a tester performs testing and the Programmer performs the coding phase, then examining all the process role involved in the the studied processes we succeeded in detailing each of these higher level stakeholder in the following:
System Analyst: models the current system and generates information about the future system, he is responsible of detailing use cases and he is an expert of the development

domain thus identifying and modeling the main elements of the multi agent system under construction (referring to the MMM elements).
Domain Analyst: analyzes the system environment in order to determine and model MAS domain elements.
User: defines and validates the system requirements.
Agent Analyst: analyzes the system to be in order to establish which entities can be agents and to research which architecture is necessary to build the system.
Agent Designer: analyzes and designs all the MMM elements strictly related to the concept of agent (in each design process), such as role, task, services, interaction language and so on.
User Interface Designer: identifies and defines the interface among actors and the system.
Programmer: is responsible of writing the code.
Test Designer: designs a test activity basing on system requirements an agent has to satisfy.
Test Developer: executes the designed tests.
Again we can see that some process roles are classic ones for object oriented context while some others, for instance agent designers, are specific stakeholders of the agent oriented context.

3.3 Work Product Kind

A generic work product produced by a process activity can be of a certain kind representing a specific category, for instance text document, code an so on; we clustered all the possible work product kinds under two main categories (Fig. 3): graphical and textual.

A work product which kind is graphical can be furthermore categorized as a structural or a behavioural one, when used to model respectively the static or the dynamic aspect of a system; for instance a behavioural work product points out the flow of messages along the time among different agents.

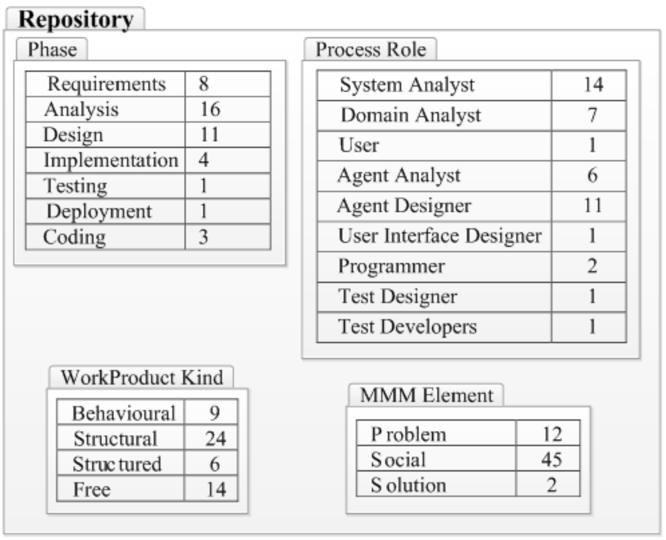
As regard the textual work product we decided to classify them as structured or free, in the sense that a text document can be hold by a particular template or grammar, for example to build a table or to write a code document, or can be freely written in a natural language.

Sometimes some work products can combine two different kinds (this is the case of a document including both a diagram and the related description) so we introduced the term atomic or composite to mean a work product of a single kind or a work product of two or more combined kinds.

3.4 MAS Model Elements

The MAS Metamodel gives a structural representation of the concepts belonging to the system under construction; in our previous works [7][8] we divided the metamodel of the multi agent system in three areas, to better deal with the different domain abstractions relevant to a system design; the first area represents all the aspects of the user's problem description including the environment representation, the second deals with agent based concepts that are useful to define a solution strategy and the third describes the structure of the code solution.

We claim that all the existing MAS Metamodels can be divided in the cited three areas thus allowing the creation of three categories of MMM elements; we named them: Problem, Social and Solution.


Repository

Phase

Requirements	8
Analysis	16
Design	11
Implementation	4
Testing	1
Deployment	1
Coding	3

Process Role

System Analyst	14
Domain Analyst	7
User	1
Agent Analyst	6
Agent Designer	11
User Interface Designer	1
Programmer	2
Test Designer	1
Test Developers	1

WorkProduct Kind

Behavioural	9
Structural	24
Structured	6
Free	14

MMM Element

Problem	12
Social	45
Solution	2

Figure 4. Repository Content

4 Overview on our repository content

Our work starts from a re-engineering activity of the analyzed processes [8][9][10][12], that let us to represent all of them in a standardized way using SPEM and to extract forty-five fragments, each of them stored in our repository on the basis of the process elements and the MMM elements categories it deals with; Fig. 4 summarizes how much fragments we stored in each category.

Some repositories of fragments already exist [11][15], in our work we propose a repository structured with the aim of minimizing the effort necessary for finding the best fragment for a specific purpose and a specific application context.

In building our repository we adopted an approach that is someway similar to the OPF one [11](we reengineered the studied processes, expressed them in a standardized notation, and then extracted the fragments); in so doing we adopted a specific choice at the basis of the extraction process: we looked at the work products as the beam for split-

ting down the process in its constituting method fragments. The result is a relatively small number of method fragments but we think this could be the right level of granularity for our purposes because, in this way, the process construction can be lead by a concrete and tangible entity, the work product, and finally it results in an easier and faster extraction and selection of fragments.
We already made an experiment on constructing a new design process using some method fragments stored in the repository; it consisted in building an agile process for rapid prototyping of applications in our laboratory. The result was the Agile PASSI [5] process that was largely based on PASSI fragments, because we wanted to reuse the expertise we accumulated in several years of using it.

The repository we presented can be accessed from a website [1] that allows the user to query the method base looking for matches on several keys from the categories we proposed in section 3; in the repository the method fragments are stored in form of text documents and the metadata are managed using a relational data structure, some relationships are, for instance, the following: *Fragment(ID,FragName,FileName,ID_Phase)* contains the data identifying each fragment with the link to its representing document and the phase it belongs to, *Phase(ID,PAName,PADescription)* contains the information on phases and in the same way for all the other elements.

5 Conclusion

In this paper we presented a repository of method fragments that can be used for composing a new design process for multi-agent systems. The peculiarities of this repository can be summarized as follows: (i) this is a specifically agent-oriented conceived repository; as a consequence a specific attention has been given to agent-oriented peculiarities like the MAS Metamodel elements that are explicitly present in both the method fragments descriptions and in their categorization; (ii) fragments have been defined (and extracted from existing methodologies) following a work product-based approach; we mean that each method fragment is supposed to produce at least one work product (not necessarily from scratch, it can also refine an existing one); (iii) we adopted a specific philosophy for enabling fragments retrieval from the method base: fragments are categorized according to 4 basic criteria: process roles involved in the design activities, phase of the overall design process in which the specific fragment can be reused, kind of work product produced by the repository and finally, MAS metamodel elements that are managed in the activities involved in the fragment. In the future we plan of extending the repository by including some other methodologies (we are currently working on Ingenias and Prometheus) and then incorporating that in a CAME/CASE tool we are building in order to effectively support the work of the process designer first (while he defines the new process) and the system designer later (while he designs the agent system).

[1] http://www.pa.icar.cnr.it/passi/fragment.html

References

[1] Bergenti, F., Gleizes, M.P., Zambonelli, F.: Methodologies and Software Engineer- ing for Agent Systems. Kluwer (2004)

[2] Boehm, B.: A Spiral Model of Software Development and Enhancement. IEEE Computer, Vol. 21, N 5, May, (1988) pp. 61-72

[3] Brinkkemper, S.: Method engineering: engineering of information systems development methods and tools. Information and Software Technology. 38(7): p. 275-280 (1996)

[4] Castro, J., Kolp, M., Mylopoulos, J.: Towards requirements-driven information systems engineering: the tropos project. Inf. Syst. 27 (2002) 365389

[5] Chella, A. Cossentino, M., Sabatucci, L., Seidita, V.: Agile PASSI: An Agile Process for Designing Agents. International Journal of Computer Systems Science & Engineering. Special issue on ”Software Engineering for Multi-Agent Systems”. May 2006. in printing

[6] Cossentino, M.: From requirements to code with the PASSI methodology. In Henderson-Sellers, B., Giorgini, P., eds.: Agent-Oriented Methodologies, Idea Group Inc. (2005)

[7] Cossentino, M., Sabatucci, L., Seidita, V.: Method Fragments from the PASSI process. Technical Report ICAR-CNR n. 21-03 (2003)

[8] Cossentino, M., Sabatucci, L., Seidita, V.: SPEM description of the PASSI process. Technical Report ICAR-CNR n. 20-03 (2003) Available on line at http://www.pa.icar.cnr.it/cossentino/FIPAmeth/metamodel.htm.

[9] Cossentino, M., Seidita, V.: SPEM Description of ADELFE Process. Technical Report ICAR-CNR n.05-07 (2005)

[10] Cossentino, M., Seidita, V.: Tropos: Processo e frammenti. Technical Report ICAR-CNR n.05-06 (2005)

[11] Firesmith, D. and Henderson-Sellers, B.:The OPEN Process Framework - An Introduction. Addison-Wesley: Harlow, UK (2002)

[12] Garro, A., Turci, P.: Gaia Fragments. available on line at http://www.pa.icar.cnr.it/cossentino/FIPAmeth/metamodel.htm

[13] Ghezzi, C., Jazayeri, M., and Mandrioli, D.: Fundamentals of Software Engineering. Prentice Hall International, Upper Saddle River, NJ (USA) (1991)

[14] Harmsen A.F.: Situational Method Engineering. Moret Ernst & Young (1997)

[15] Harmsen,A.F., Brinkkemper, S.: Design and Implementation of a Method Base Management System for a Situational CASE Environment. APSEC 1995: 430-438

[16] Harmsen A.F., Brinkkemper, S., Oei, H.: Situational Method Engineering for Information System Projects. In Olle T.W. and A.A. Verrijn Stuart (Eds.), Mathods and Associated Tools for the Information Systems Life Cycle, Proc. of the IFIP WG8.1 Working Conference CRIS'94, pp. 169-194, North-Holland, Amsterdam, (1994)

[17] Henderson-Sellers, B.: Process Metamodelling and Process Construction: Examples Using the OPEN Process Framework (OPF). Ann. Softw. Eng. 14, 1-4, 341-362 (Dec. 2002)

[18] Kumar K., Welke R.: Methodology engineering: a proposal for situation-specific methodology construction. In Challenges and Strategies for Research in Systems Development, pages 257269, 1992.

[19] Method fragment definition. FIPA Document, http://www.fipa.org/activities/methodology.html, (Nov 2003)

[20] OMG, 2002, Software Process Engineering Metamodel Specification, Version 1.0, Object Management Group, formal/02-11-14 (Nov 2002)

[21] Ralyté, J.: Towards situational methods for information systems development: engineering reusable method chunks, Procs. 13th Int. Conf. on Information Systems Development. Advances in Theory, Practice and Education Vilnius Gediminas Technical University, Vilnius, Lithuania, 271-282 (2004)

[22] Saeki, M.: Software Specification & Design Methods and Method Engineering. International Journal of Software Engineering and Knowledge Engineering.

[23] Sommerville, I.: Software Engineering. Addison-Wesley (2004)

[24] Zambonelli, F., Jennings, N., Wooldridge, M.: Developing multiagent systems: the gaia methodology. ACM Transactions on Software Engineering and Methodology 12 (2003) 417470

Expressing preferences declaratively in logic-based agent languages

Stefania Costantini
Università degli Studi di L'Aquila
Dipartimento di Informatica
Via Vetoio, Loc. Coppito,
I-67010 L'Aquila - Italy
Email: stefcost@di.univaq.it

Pierangelo Dell'Acqua
Department of Science and Technology - ITN
Linköping University
601 74 Norrköping, Sweden
Email: pier@itn.liu.se

Arianna Tocchio
Università degli Studi di L'Aquila
Dipartimento di Informatica
Via Vetoio, Loc. Coppito,
I-67010 L'Aquila - Italy
Email: tocchio@di.univaq.it

Abstract— **In this paper we present an approach to introducing preferences among actions in logic-based agent-oriented languages. These preferences are expressed in the body of rules (i.e., they are local to the rule where they are defined). To the best of our knowledge, no similar approach has been proposed before, and cannot be easily simulated by means of preferences expressed in the head of rules, which are global. The approach is applied to choosing which action an agent should perform in reaction to an event, among the feasible ones.**

I. INTRODUCTION

Intelligent agents perform advanced activities such as negotiation, bargaining, etc. where they have to choose among alternatives. The choice will be based on some kind of preference or priorities related for instance to:

- the agent's objectives;
- the context (cooperative vs. competitive);
- available resources;
- the strategies that the agent intends to follow.

Agents will in general include specialized modules and/or meta-level axioms for applying priorities and preferences, like for instance those proposed in [9] for prioritized defeasible reasoning. However, it can be useful in logical agents to be able to express preferences at a more basic linguistic level. These basic preferences can then be employed in building more advanced high-level strategies. At the language level, preferences have already been expressed in various way in Answer Set Programming [8] [12]. In that context, the basic mechanism is that of computing the Answer Sets and then chose "preferred" ones. We will shortly review below the work of Brewka on LPODS (Logic Programs with Ordered Disjunction) [2] and the work of Sakama and Inue on PLP (Prioritized Logic Programming) [13]. The reader may refer to the latter paper and to [6] for a discussion of relevant existing approaches. Some of them are based on establishing priorities/preferences among atoms (facts), and typically introduce some form of disjunction in the head of rules. Other approaches express instead priorities among rules.

Our proposal is aimed at allowing an agent to express preferences concerning either which action they would perform in a given situation, or, in perspective, which goal they would pursue in a certain stage. Since actions are often performed in reaction to events and goals are set in accordance to some internal conclusion that has been reached, we propose to introduce disjunction in the body of rules. If the body of a rule contains a disjunction among two or more actions, the *preferred* one will be chosen according to *preference rules*, that may have a body in that (following [13]) priorities may be conditional. If agents evolve with time and enlarge their knowledge and experience, priorities may dynamically change according to the agent evolution.

In agent languages that are not based on Answer Set Programming, one cannot select the preferred model(s) by means of a filter on possible models. Then, other techniques are needed in order to provide a semantic account to this proposal. Recently, an approach to declarative semantics of logical agent-oriented languages that considers evolution of agents has appeared in the literature [5]: changes that occur either externally (i.e., reception of exogenous events) or internally (i.e., courses of actions undertaken based on internal conditions) are considered as making a change in the agent program, which is a logical theory, and in its semantics (however defined). For such a change to be represented, it is understood as the application of a program-transformation function. Thus, agent evolution is seen as program evolution, with a corresponding semantic evolution.

This semantic approach can be applied to the present setting by adapting the proposal of the *split programs* introduced in [2]. A split program is a version of the given program obtained by replacing each disjunction by one of its options. Then, at each step we would have a set of *possible evolutions*, each corresponding to a split program. Among them, the preferred one (according to the present conditions) is taken, while all the others are pruned. As mentioned before, given similar situations in different stages of the agent life, different options can be taken, according to the present assessment of the agent knowledge.

Though simple, this mechanism is to the best of our knowledge new, as no similar approach has been proposed before, and it cannot be easily simulated by existing ones.

In Section II we review some features that intelligent logical agents should in our opinion possess, and the related usefulness of introducing preferences. In Section III we briefly review previous related work on preferences. In Section V we introduce the approach, and in Section VI its semantics. Finally, we conclude in Section VII.

II. Enhancing capabilities of Logical Agents by introducing Preferences

A great deal can be said about features that agents in general and logical agents in particular should possess (for a review the reader may refer for instance to [14], for a discussion to [11]). It is widely recognized however that agents, whatever the language and the approach on which they are based, should be able to cope with a changing and partially known environment. In this environment, agents should be able to interact, when they deem it appropriate, with other agents or with the user in order to complete their own problem solving and to help others with their activities.

Interacting agents may act according to suitable strategies, which include expressing preferences and establishing priorities, possibly with the aid of past experiences. In our view, complex strategies can take profit of basic linguistic constructs reminiscent of those introduced in Answer Set Programming.

Our proposal is aimed at allowing an agent to express preferences/priorities (in the following, we will often interchange the two terms) concerning either which action they would perform in a given situation, or also, in perspective, which goal they would pursue in a certain stage. Since actions are often performed in reaction to events and goals are set in accordance to some internal conclusion that has been reached, we propose to introduce disjunction in the body of rules. If the body of a rule contains a disjunction among two or more actions, the *preferred* one is chosen according to *preference rules,* that may have a body. I.e., following [13], priorities may be conditional. Also, preference rules may contain references to the agent past experience, and then the preferred choice may change over time. More precisely, whenever the body of a rule contains a disjunction among two or more actions, the intended meaning is the following:

- preference rules establish which action is preferred;
- precondition of the action state whether it can be actually performed, i.e., if it is feasible;
- the agent should perform the best preferred feasible action.

III. Previous Related Work

The reader may refer to [6] for a discussion of many existing approaches to preferences. The main distinction is among those that define priorities/preferences among atoms (facts), and typically introduce some form of disjunction in the head of rules, and those that express instead priorities among rules. Among the latter ones, we mention [10] that applies preferences among rules in negotiating agents based on argumentation, so as to tune argumentations according to changing contexts.

The approach of [13] considers *general extended disjunctive programs* where a rule has the syntax:

$$L_1| \dots |L_k| not\, L_{k+1}| \dots |not\, L_{k+h} \leftarrow Body$$

where "|" represents disjunction and not is negation as failure under the Answer Set semantics. A preference, or priority, between two ground literals e_1, e_2 is expressed in the form $e_1 \prec e_2$. An answer set S_2 of a given program is preferable onto another answer set S_1 iff $S_2 \setminus S_1$ contains an element e_2 whose priority is higher than some element e_1 in $S_1 \setminus S_2$, and the latter does not contain another element e_3 whose priority is strictly higher that e_2. Then, preferred answer sets (or p-answer sets) are a subset of the traditional ones, that can be seen as a special case corresponding to empty priorities.

Basic PLP is exploited in [13] so as to express priorities not only between plain facts, but also between more general forms of knowledge. The approach allows many forms of commonsense reasoning to be modeled.

An interesting application is that of *priority with preconditions*. For instance, borrowing the example from [13], the situation where a person drinks tea or coffee but she prefers coffee to tea when sleepy can be represented as follows (in a prolog-like syntax):

$$tea \mid coffee.$$
$$tea \prec coffee \text{ :- } sleepy.$$

This program can be translated in a standard way in plain PLP and, assuming that *sleepy* holds, has the p-answer set $\{sleepy, coffee\}$.

In LPODS [2], one can write expressions such as $A \times B$ in the head of rules, where the new connective $\times$ stands for ordered disjunction. The expression intuitively stands for: if possible A, but if A is impossible then (at least) B. If there are several disjuncts, the first one represents the best preferred option, the second one represents the second best option, etc. The following is an example where a person who wishes to spend the evening out and has money prefers to go to theatre, or else (if impossible) to go to the cinema, or else (if both previous options cannot be taken) to go to dine at a restaurant.

$$theatre \times cinema \times restaurant \text{ :- } want\_to\_go\_out, have\_money.$$

For selecting the preferred answer set(s) of a program P, one obtains the possible split programs of P, where a split program P' is obtained from P by replacing each disjunctive rule by one of its options. Then, the answer sets of P are taken to be the answer sets of the split programs. To choose preferred ones given that there may be several disjunctions, a notion of *degree of satisfaction* of disjunctive rules must be defined, that induces a partial ordering on answer sets. Preferred answer sets are those that satisfy all rules of P to the better degree.

IV. Comparison

To the best of our knowledge, the approach of introducing preferences in the body of logical rules is novel, and has never appeared in the literature. It cannot be easily simulated by using preferences in the head: in fact, preferences expressed in the body are *local* to the rule where they occur, while preferences defined in the head are *global*. The application to agents performing actions is also new. As an agent evolves in time and its knowledge changes, preferred choices will change as well. Then, according to the same preference structure an agent will in general prefer differently in different stages of its life.

V. The approach in more detail

We will now introduce a simple though in our opinion effective construct that can be employed in agent-oriented logic languages based on logic (horn-clause) programming. Similarly to [13], we assume the following:

- preferences are expressed between two ground facts;
- preferences are expressed explicitly by means of special rules, that may have conditions;
- preference is transitive, irreflexive and anti-symmetric.

In our approach, preferences can be defined between *actions* that agents may perform. We make some preliminary assumption about the agent languages we are considering. We do not commit to any particular syntax, though we will propose a sample one in order to introduce and illustrate examples. We will discuss the semantics of the class of languages that we consider in Section VI. By saying "an agent" we mean a program written in the language at hand, that behaves as an agent when it is put at work. We assume in particular the following syntactic and operational features.

- The agent is able to perceive external events coming from the environment where the agent is situated. In our sample syntax an external event is an atom which is distinguished by postfix E. E.g., $rainE$ indicates an external event.
- The agent is able to react to external events, i.e., the language provides some kind of condition-action construct. In our sample syntax ee indicate reaction by means of the connective $:>$. Then, a reactive rule will be indicated with $pE :> Body$ meaning that whenever the external event pE is perceived, the agent will execute $Body$. There are languages (like, e.g., the one presented in [3]) where an agent can react to its own internal conclusions, that are interpreted as events (thus modeling proactivity). We assume that the syntax for reaction is the same in both cases. However, an internally generated event is indicated with postifix I, i.e., in the form pI.
- The agent is able to perform actions. Actions will occur in the agent program as special atoms. In our sample syntax we assume them to be in the form qA, i.e., they are distinguished by suffix A. E.g., $open\_umbrellaA$ indicates an action. Actions may have preconditions: In our sample syntax we assume them to be expressed by rules. The connective $:<$ indicates that the rule defines the precondition of an action. I.e., a precondition rule will be indicated as $qA :< Body$, meaning that the action qA can be performed only if $Body$ is true. We do not cope here with the effective execution of actions, that is left to the language run-time support.

In the proposed approach, a disjunction (indicated with "|") of actions may occur in the body of a reactive rule. Preferences among actions are defined in *preference rules*, that are indicated by the new connective $<<$. Then, a rule $pE :> q1A \mid q2A$ means that in reaction to pE the agent may perform either action $q1A$ or action $q2A$. A rule $q1A << q2A$:- $Body$ means that action $q2A$ is preferred over action $q1A$ provided that $Body$ is true. I.e., if $Body$ is not true the preference is not applicable, and then any of the actions can be indifferently executed. A set of preference rules define in general a *partial order* among actions, where preferences are transitively applied and actions that are unordered can be indifferently executed. In our approach preferences are applied on *feasible* actions. I.e., the partial order among actions must be re-evaluated at each step of the agent life where a choice is possible, according to the preconditions of the actions. The preferred actions at each stage are those that can actually be performed and that are selected by the preference partial order.

Example 5.1: Consider a person who receives an invitation to go out. She would prefer accepting the invitation rather than refusing, provided that the invitation comes from nice people. She is able to accept if she has money and time. The invitation is an *external event* that reaches the agent from her external environment. Accepting or refusing constitutes the *reaction* to the event, and both are actions. One of the actions (namely, accepting) has preconditions. In our sample syntax, an agent program fragment formalizing this situation may look as follows.

$$invitationE :> acceptA \mid rejectA.$$
$$acceptA :< have\_money, have\_time.$$
$$refuseA << acceptA \text{ :- } nice\_people\_inviting.$$

When the external event $invitationE$ is perceived by the agent, it can react by alternatively performing one of two actions. The action $acceptA$ will be performed if its preconditions are verified. As preferences are among feasible actions, $acceptA$ is preferred provided that $nice\_people\_inviting$ holds. Notice that this is not known in advance, as the agent evolves in time: the invitation may arrive at a stage of the agent operation when time and money are available, and then the preferred action is chosen. If instead the invitation (or, another future invitation) arrives when there are no resources for accepting, the agent will refuse the invitation.

Another example will introduce further aspects.

Example 5.2: Let us now rephrase the example of the person preferring coffee over tea if sleepy. Let us put it in a proactive perspective, where the person wonders whether it is time to take a break from working, e.g., at mid-afternoon. If so, she will consider whether to drink tea or coffee. The corresponding program fragment might look as follows, where $take\_break$ is an *internal conclusion* that triggers a proactive behavior: the first rule reaches the conclusion that taking a break is in order; the second rule states what to do then, i.e., specifies a reaction to the internal conclusion itself (indicated in the second rule with postfix I for "internal"). For the mechanism to be effective, $take\_break$ must be attempted from time to time, so as to trigger the consequent behavior as soon as it becomes true.

$take\_break$:- $five\_oclock$.
$take\_breakI$:> $drink\_teaA \mid drink\_coffeeA$.
$drink\_coffeeA$:< $espresso$.
$drink\_teaA << drink\_coffeeA$:- $sleepy$.

Again, what the agent will do depends upon the present conditions, i.e., upon whether the agent feels sleepy or not. Moreover, in this variation the agent drinks coffee only if she can have an espresso.

Assume now that there is also the option of drinking juice, though the agent will only drink orange juice, and that the agent prefers juice to tea. Then the program becomes:

$take\_break$:- $five\_oclock$.
$take\_breakI$:> $drink\_teaA \mid drink\_coffeeA$
$\mid drink\_juiceA$.
$drink\_coffeeA$:< $espresso$.
$drink\_juiceA$:< $orange$.
$drink\_teaA << drink\_coffeeA$:- $sleepy$.
$drink\_teaA << drink\_juiceA$.

The expected behavior is the following:

- If $sleepy$ holds and $espresso$ holds as well, the agent can drink coffee (the action $drink\_coffeeA$ is allowed) and will not drink tea, which is less preferred. If $orange$ does not hold, the agent will definitely drink coffee.
- If $sleepy$ holds and $espresso$ holds as well, the agent can drink coffee (the action $drink\_coffeeA$ is allowed) and will not drink tea, which is less preferred. If $orange$ holds, also the action $drink\_juiceA$ is allowed, and preferred over $drink\_teaA$. The agent can indifferently drink either coffee or juice, as they are unrelated.
- If $espresso$ does not hold, the agent cannot drink coffee (the action $drink\_coffeeA$ is not allowed). Then, if $orange$ holds then the agent will drink juice (the action $drink\_juiceA$ will be performed), otherwise it will drink tea (as the action $drink\_teaA$ is always allowed, not having preconditions).
- If $sleepy$ does not hold, there is no preference between tea and coffee. If $orange$ does not hold and $espresso$ holds, one of the two actions $drink\_teaA$ or $drink\_coffeeA$ can be indifferently executed. If $orange$ holds and $espresso$ holds as well, $drink\_juiceA$ is preferred over $drink\_teaA$, but as no other priority is specified, one of the actions $drink\_coffeeA$ or $drink\_juiceA$ can be indifferently executed.

VI. Declarative Semantics of Evolving Agents with Preferences

The evolutionary semantics that has been proposed in [5] has the objective of providing a unifying framework for various languages and semantics for reactive and proactive logical agents.

This semantic approach is based upon declaratively modeling the changes inside an agent which are determined both by changes in the environment and by the agent's own self-modifications. The key idea is to understand these changes as the result of the application of program-transformation functions. In this view, a program-transformation function is applied for instance upon reception of either an external or an internal event, the latter having a possibly different meaning in different formalisms. That is, perception of an event can be understood as having an effect on the program which defines the agent: for instance, the event can be stored as a new fact in the program. Similarly, actions which are performed can be recorded as new facts. All the "past" events and actions will constitute the "experience" of the agent.

Recording each event or action or any other change that occurs inside an agent can be semantically interpreted as transforming the agent program into a new program, that may procedurally behave differently than before: e.g., by possibly reacting to the event, or drawing conclusions from past experience. Or also, the internal event corresponding to the decision of the agent to undertake an activity triggers a more complex program transformation, resulting in version of the program where the corresponding *intention* is somewhat "loaded" so as to become executable.

Then, in general one will have an initial program P_0 which, according to these program-transformation steps (each one transforming P_i into P_{i+1}), gives rise to a Program Evolution Sequence $PE = [P_0, ..., P_n]$. The program evolution sequence will have a corresponding Semantic Evolution Sequence $ME = [M_0, ..., M_n]$ where M_i is the semantic account of P_i according to the specific language and the chosen semantics. The couple $\langle PE; ME \rangle$ is called the *Evolutionary Semantics* of the agent program P_{Ag}, corresponding to the particular sequence of changes that has happened, and to the order in which they have been considered. The evolutionary semantics of an agent represents the history of an agent without introducing a concept of a "state".

The different languages and different formalisms will influence the following key points:

1) When a transition from P_i to P_{i+1} takes place, i.e., which are the external and/or internal factors that determine a change in the agent.
2) Which kind of transformations are performed.
3) Which semantic approach is adopted, i.e., how M_i is obtained from P_i. M_i might be for instance a model,

or an initial algebra, or a set of Answer Sets if the given language is based on Answer Set Programming (that comes from the stable model semantics of [8]). In general, given a semantics $\mathcal{S}$ we will have $M_i = \mathcal{S}(P_i)$.

A transition from P_i to P_{i+1} can reasonably take place, for instance:

- When an event happens.
- When an action is performed.
- When a new goal is set.
- Upon reception of new knowledge from other agents.
- In consequence to the decision to accept/reject the new knowledge.
- In consequence to the agent decision to revise its own knowledge.

We say that at stage P_{i+1} of the evolution the agent *has perceived* event ev (whatever its class) meaning that the transition from P_i to P_{i+1} has taken place in consequence of reception of ev. It is reasonable to assume that in the stage P_{i+1} the agent will cope with ev, e.g., by reacting to it if it is an external event.

Example 6.1: It is useful to discuss how the program transformation step related to actions might be formalized. Intuitively, an action atom (like e.g. $drink\_coffeeA$ in a previous example) should become true given its preconditions, if any ($espresso$ in the example) whenever the action is actually performed in some rule. For the sake of simplicity assume that (like in the examples presented above) actions can occur only in the body of reactive rules.

Declaratively, this means that the action occurs in the body of an applicable reactive rule. Practically, whenever that rule will be processed by the interpreter because the corresponding (external or internal) event has happened, the action will be actually performed (by means of any kind of mechanism that connects the agent to its environment). To account for this behavior, in the initialization step each rule defining preconditions for actions, say of the form

$actA :< C_1, \ldots, C_s$

is transformed into a set of rules of the form:

$actA$:- $D_1, \ldots, D_h, C_1, \ldots, C_s$

where $D_1, \ldots, D_h$, $h \geq 0$ are the conditions (except for other actions) of each reactive rule where $actA$ occurs in the body. The C_i's are omitted if $actA$ has no preconditions.

Whenever at some stage P_i of the program evolution $actA$ will be attempted and feasible as its preconditions are true, we will have $actA \in M_i$.

It can be useful in general to perform an *Initialization step*, where the program P_{Ag}, written by the programmer, is transformed into a corresponding initial program P_0 by means of some sort of knowledge compilation. This initialization step can be understood as a rewriting of the program in an intermediate language and/or as the loading of a "virtual machine" that supports language features. This stage can on one extreme do nothing, on the other extreme it can perform complex transformations by producing "code" that implements language features in the underlying logical formalism. P_0 can be simply a program (logical theory) or can have additional information associated to it.

This semantic approach can be extended so as to encompass the present proposal. As a first point, in the initialization step preferences must be collected and preference rules removed. Then, P_0 will not contain preference rules, but will be associated to a structure $Pref$ where preferences between couples of (ground) actions are made explicit, by performing the transitive closure of preference rules. The conditions of a preference rule (if any) are added as preconditions of the preferred action.

We adapt the idea of *split program* from [2]. A split program is a version of the given program obtained by replacing a disjunction by one of its options. In our case, whenever an agent at stage P_i of its evolution has perceived an (either external or internal) event, say pE, it will react to it. However, if there is a disjunction of actions in the body of the corresponding reactive rule, then the agent may react in more that one way. The different ways of reacting are represented by different *split programs*, each one representing an alternative. Precisely,

Definition 1: Let P_{Ag} be an agent program that has been transformed into a program P_0 by the initialization step. Let P_i be the program obtained from the evolution of P_0 at the i-th step, corresponding to the perception of event pE. Let $pE :> Body$ be the corresponding reactive rule in P_i, where a disjunction of actions occurs in $Body$. A *split program* P_i' is obtained by replacing the disjunction with one of its options.

Referring to the program of Example 5.1, at the initialization step it is transformed into:

$$
\begin{array}{l}
invitationE :> acceptA \mid rejectA. \\
acceptA :< have\_money, have\_time, \\
\qquad\qquad nice\_people\_inviting.
\end{array}
$$

where the preference $refuseA << acceptA$ is recorded in the structure $Pref$. Then, whenever the event $invitationE$ will be perceived will be two split programs: a first one, say ϕ_1, where the body of the reactive rule contains only $acceptA$, and a second one, say ϕ_2, where the body of the reactive rule contains only $refuseA$.

We will have a set $\{P_i^1, \ldots, P_i^k\}$ of split programs corresponding to the number k of actions occurring in the disjunction. Assuming that events are considered one at a time (i.e., an evolution step copes with a single event), at each stage split programs will be relative to a single reactive rule, and will correspond to a set $\{M_i^1, \ldots, M_i^k\}$ where M_i^j is the semantics of P_i^j. We say that we *a split occurs* at stage P_i of program evolution whenever at that stage the incoming event is related to a reactive rule with a disjunction of actions in its body.

The preferred split programs are those whose semantics contain the preferred actions. Precisely:

Definition 2: Let P_{Ag} be an agent program that has been transformed into a program P_0 by the initialization step, and let $Pref$ be the preference structure that has been associated to the program. Let P_i correspond to a step of the evolution of P_0, where a split occurs. Given two split programs P_i^r and P_i^s obtained from P_i by splitting a disjunction $act^1 A \,|\, \ldots \,|\, act^k A$, then P_i^r is preferred over P_i^s if the following conditions hold:

- the semantics M_i^r of P_i^r contains $act^r A_i$;
- $act^r A_i$ is preferred over $act^s A_i$ according to $Pref$.

Notice that both M_i^r and M_i^s may not contain the corresponding action ($act^r A_i$ and $act^s A_i$ respectively), in case its preconditions are false. Then, a split program is preferred upon another one if (i) its semantics entails the related action and (ii) either the semantics of the other one does not entail the related action, or the former action is preferred.

Then, at each step where a split occurs we have a set of *possible evolutions*, each corresponding to a split program. Among them, the preferred one (according to the present conditions) is taken, while all the others are pruned. As mentioned before, given similar situations at different stages of the agent life, different options can be taken, according to the present assessment of the agent knowledge. In the example, ϕ_1 will be preferred to Φ_2 whenever it actually entails $acceptA$.

We can have a unique best preferred split program P_i^b if $Pref$ is a total order with respect to the actions over which we split, or we may have more than one equally preferred split programs. Any of them can be indifferently selected.

Definition 3: Let P_i be a stage of the program evolution sequence, where a split occurs. We let P_{i+1} be any of the best preferred split programs.

VII. Concluding Remarks

In this paper we have presented an approach to expressing preferences among actions and in logical agents. The approach builds on previous relevant work related to answer set programming, but is rephrased for reactive and proactive agents that evolve in time. In fact, the semantics of the approach is given in *evolutionary* terms, where an agent program is considered to be modified by events that happen and actions that are performed, while its semantic account evolve correspondingly.

There are others approaches in computational logic that are related to the present one, and to which we are indebted, namely [1] and [7], where preferences and updating preferences are coped with in the context of a more general approach to updating logic programs. We may notice that the examples that we have presented basically refer to the syntax and procedural semantics of the DALI language [3], [4], [14]. Actually they correspond to working DALI programs, though the implementation is prototypical and is being experimented.

Our next research objective is to extend the possibility of expressing preferences to all kinds of subgoals occurring in the body of logical rules, instead of coping with actions only.

Another important objective is to extend the approach so as to be able to express preferences among agent goals (objectives to reach). In fact, the reasoning can be similar. Actually, setting an objective is related to building a plan for achieving it. A plan however can be seen as divided into:

1) a preliminary check stage, where feasibility of subsequent actions is checked (are the tickets available? Do I have the money? Do my friends accept to join me? May I rent a car?);
2) an operative stage, where actions that influence the environment (and in general cannot be retracted, or at least not so easily) are performed.

The first stage can be seen as a feasibility stage for setting an objective. Then, if there is a disjunction of objectives in the body of a rule, we mean that the agent should set the most preferred feasible one.

References

[1] J. J. Alferes, P. Dell'Acqua and L. M. Pereira, *A compilation of updates plus preferences*. Logics in Artificial Intelligence, Proc. of the 8th Europ. Conf., JELIA 2002, LNAI 2424, Springer-Verlag, 2002, pp. 62-74.

[2] G. Brewka, *Logic programming with ordered disjunction*, In Proc. of AAAI-02, Edmonton, Canada, 2002.

[3] S. Costantini and A. Tocchio, *A logic programming language for multi-agent systems*. Logics in Artificial Intelligence, Proc. of the 8th Europ. Conf., JELIA 2002, LNAI 2424, Springer-Verlag, 2002.

[4] S. Costantini, A. Tocchio, *The DALI logic programming agent-oriented language*. Logics in Artificial Intelligence, Proc. of the 9th European Conference, Jelia 2004, Lisbon, September 2004. LNAI 3229, Springer-Verlag, Germany, 2004.

[5] S. Costantini, A. Tocchio, *About declarative semantics of logic-based agent languages*. In M. Baldoni and P. Torroni (eds.), Declarative Agent Languages and Technologies, Post-Proc. of DALT 2005. LNAI 3229, Springer-Verlag, Germany, 2006.

[6] J. Delgrande, T. Schaub, H. Tompits and K. Wang, *A classification and survey of preference handling approaches in nonmonotonic reasoning*. Computational Intelligence, 2004.

[7] P. Dell'Acqua and L. M. Pereira, *Preferring and updating in logic-based agents*. In: Web-Knowledge Management and Decision Support. Selected Papers from the 14th Int. Conf. on Applications of Prolog (INAP), LNAI 2543, Springer-Verlag, Berlin, 2003, pp. 70-85.

[8] M. Gelfond and V. Lifschitz, *The stable model semantics for logic programming*. Proc. of 5th ILPS conference, 1988, pp. 1070-1080.

[9] M. Gelfond and T. C. Son, *Reasoning with prioritized defaults*. In Proc. of the 3rd Int. Works. on Logic Programming and Knowledge Representation, LNAI 1471, Springer-Verlag, Berlin, 1998, pp. 164-223.

[10] A. Kakas and P. Moraitis, *Adaptive agent negotiation via argumentation*. In Proc. of the 5th International Joint Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS'06), Hakodate, Japan, 2006.

[11] R. A. Kowalski, *The logical way to be artificially intelligent, Computational Logic in Multi-Agent Systems* (CLIMA VI Post-Proceedings), LNAI 3900, Springer-Verlag, Berlin, 2005, pp. 1-22.

[12] V. Lifschitz, *Answer set planning.* Invited talk. Proc. of ICLP '99 Conf., pp. 23–37. The MIT Press, 1999.

[13] C. Sakama and K. Inoue, *Prioritized logic programming and its application to commonsense reasoning*. Artif. Int. 123(1-2), Elsevier, 2000, pp. 185-222.

[14] A. Tocchio, *Multi-Agent systems in computational logic*, Ph.D. Thesis, Dipartimento di Informatica, Universitá degli Studi di L'Aquila, 2005.

Models, Abstractions and Phases in Multi-Agent Based Simulation

Mizar Luca Federici, Stefano Redaelli and Giuseppe Vizzari
Dept. of Computer Science, Systems, and Communication
University of Milan–Bicocca
Milan, Italy
{mizar,redaelli,vizzari}@disco.unimib.it

***Abstract*— This paper introduces some considerations about simulation practice and a schema of the models that are implicitly and explicitly involved in a Multi-Agent Based Simulation (MABS). The aim of this work is to set simulation inside scientific framework. In order to do that we give an interpretation of the levels that compound a simulation and that constitute different kinds of abstraction. A clear awareness of the relations that exist between these levels and the corresponding steps, in fact, it is necessary if MABS wants to be adopted as a scientific investigation method. Our opinion is that this analysis suggests some answers to the objections that are often directed towards the use of simulation in scientific practice but also underlines some criticalities in this process.**

I. Introduction

Why do we use computer simulations? Winsberg in [1] claims that "simulations are often performed to investigate systems for which data are sparse". An assumption that stands behind the use of simulation as a tool to investigate reality is that reality cannot be known analyzing what can be considered as its single parts or components, but only by recreating it from its components. Assertors of the efficacy of the scientific use of simulation claim that a simulation expresses a theory, and that a theory expressed by a simulation produces a series of empiric predictions that can be directly compared to reality. This consideration lays on the conviction that a simulation is also a way to build up a scientific theory as it allows to check immediately the consequences of our assumptions and gives us precise indications for the correction or reformulation of a theory. Computer extends our calculus and memory capacity, and simulations that run on a computer are considered a method to study complex systems thanks to the possibility that they give to handle complex models. Under this extent computer simulations are also considered as virtual experimental laboratories to study phenomena that are difficult to observe directly. However there are also many objections to the use of simulations in science. Daniel Dennett in [2] asks if we can consider real motion what we see in the Cellular Automata world or if it is only apparent motion (the consideration can be referred to Multi-Agent based Simulations as well). Dennett says "The flashing pixels on the computer screen are a paradigm case, after all, of what a psychologist would call apparent motion [...] should we say at least that these moving patterns are real?". Dennett, explaining another objection to the use of Cellular Automata simulation in [2] affirms that "There has been a distinct ontological shift as we move between levels [...] whereas at the physical level there is no motion, and only individuals, cells are defined by their fixed spatial location, at design level we have the motion of persisting objects [...]". In fact when a real phenomena is studied by means of software entities that interact in a computer, it is not easy to demonstrate the correspondence of the dynamics observed in the computer to the ones that belong to a real phenomena. These are not the only objections to the "scientific" use of simulations. The introduction in a simulation of a certain amount of arbitrary details (or assumptions), that are not derived from observations, is often necessary in order to make the simulation run. This makes then difficult to understand which outputs of the simulation are really meaningful and which are instead effects of the introduction of the arbitrary details that we put into the simulation. Moreover, simulations are also criticized to be too simplified in respect to reality as it does not exist a defined criteria to guess if simplifications operated in building a simulation are the good ones. Other considerations regard the fact that simulations do not tell anything new as they are fully deterministic and just give back as output the same information that we put in input. Although many of these objections can be directed also to common scientific practice (more considerations about this topic can be found summarized in [3] [4]) it is undeniable that scientific investigation by simulation is an activity that has to be performed carefully; in fact the possibility of misinterpretation of data is higher than in common experimental practice, exactly due to the "ontological shift" mentioned by Dennett. Winsberg in [1] makes clear that many layers of models are involved in a simulation and different resources are used in each inferential step that is presumed in the shift from one layer to another. Winsberg, again in [1], explains that by these steps the simulation, from an existing theoretical knowledge, attempts to extract new knowledge about the system being simulated. This article explores the role of a specific kind of computer simulations, the MABS, in scientific investigation. In Section II-A is presented a brief overview of some definition of model in science and the relation that exists between a model and a theory. In Section II-B is introduced the scientific cycle in experimental science, in order to make, in Section III, a correlation between models in science and models in simulations. The point of view of the article is that computer simulations are not only scientific “models” of phenomena but are constituted of different layers of abstractions and present one step more in relation to scientific abstractions in experimental science. A schema of the abstractions that are implicitly involved in simulation practice is proposed and, in Section IV, we suggest that, if this activity is interpreted under this perspective, an answer can be found to some of the objections that are often moved to the use of simulation in scientific investigation. In Section V are stated the conclusions.

II. Models and Theories

A. A reflection on Models and Theories

Doran and Gilbert in [5] define a model as something that is similar to a target system *T* but easier to observe. A model *M* of a target system *T* consist in an entity that becomes the object of study in place of *T* in all the cases where this last cannot be studied or observed directly. Doran and Gilbert explain that the idea of the possibility of studying a model of *T* in place of *T* is based upon the conviction that if something is proved to be true in the model then it must be true also in the target system if, in the design of the model, some characteristics of the behavior of the target system have been “captured” by the model. This definition of model is close to the one given by Bruce Edmonds in [6]. Edmonds defines a model as something that “enables an inference process such that the process enabled in the model corresponds to some aspects of an observed process”. For this reason the result of the “inferential process” in the model, if the initial conditions are set properly, remarks the author, predicts some aspects of a subsequent state of the system that is under study. According to Edmonds’ opinion, in science a natural process is *encoded* in a model; the model performs the *inference process* and then the results of the inference are *decoded* and projected to reality in order to state a *prediction*. Edmonds in his article adds that something is a model of “something else” if the diagram in Figure 1 “commutes” and though same results follow both the lines. Under this perspective the *inference process* is thus assigned to the model. Analogously R.I.J. Huges in [7] explains that a model must be analyzed by three activities of mind that are: *denotation*, *demonstration* and *interpretation*. In Huges’ opinion, *denotation* is the “core” of the representation and consist in symbols that stand or refer to parts of the target system; *demonstration* refers to the internal dynamics of the representation (the model) whose effects can be examined; *interpretation* instead consist in the examination of the behavior of the model in order to draw conclusions on the behavior of the world. Eric Winsberg in [8] states that as “models are partially independent of both theories and the world [...] they can be used as instruments of exploration in both domains”. But for this same reason, that a model is independent both of empirical facts and of theories, the translation of the model in the target system must be accurately specified. This independent status of models is remarked also in many other works found in literature (see [9] [10]).

We have spoken about the role of models in relation to theories. But what is then a scientific theory? M.L.Dalla Chiara and G.Toraldo di Francia in [11] explain that a theory is

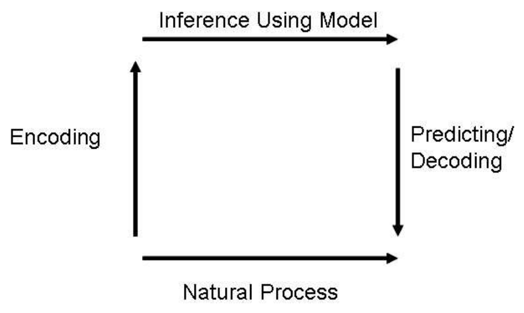


Fig. 1. Edmonds’schema representing the role of model in science

a form of knowledge that resumes a set of laws that can be applied to infinite different cases. The authors add that a form of knowledge has an axiomatic structure and from that axioms, or initial postulates, by a relation of logical consequence is established the set of derived propositions that the theory asserts. This is what is called syntactic definition of theory, but it does exists also a semantic definition stating that some kinds of theories are isomorphic to reality and that the inference steps, bringing from the set of assumptions to the consequences of the theory, are not deductive (see Winsberg [1]).

From these considerations we can infer that a model in science derives from a theory, but it is also something that corresponds to reality, although it is distinct also from it. That “correspondence”, to be taken for granted, must be made explicit although it can be proved only by empirical facts.

B. Model, Theories and Phases in Physics

Experimental science is founded on the concept of experiment and of observation of phenomena. David Hestenes in [12] asserts that “the construction of a physical model reduces a real system to an abstract model that it is possible to translate in a mathematical form”. Under his perspective a “mathematical model is at a higher abstract level and constitutes the highest abstraction of the knowledge process” (i.e. equations). The description of the observed phenomena is then constituted by the analytical or the graphic relations between sizes [12]. Axel Gelfert, in [13] states that a mathematical model, in the sense close to the one meant by Hestenes, is a set of equations, theorems and definitions but, Gelfert remarks, the equations do not constitute, by their own, a model of anything (neither of a class of phenomena) unless an interpretation that connects the variables with aspects of observable phenomena is established. We exemplify this process with the schema in Figure 2. In the same direction is also the definition given by Dalla Chiara and Toraldo di Francia that in [11] claim that a physical model can be identified with a structure $M = <Mat, Exp, Tra>$ where Mat represent the mathematical part, Exp represents the experimental part, and Tra is a function of translation that associates a mathematical interpretation to the elements of the experimental part. Axel Gelfert again in [13] underlines that mathematical models, like other kinds of models, require background assumptions for their interpretations. This

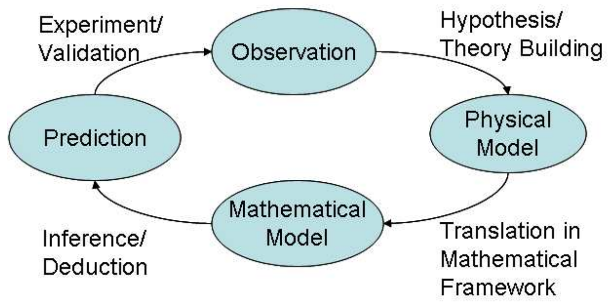


Fig. 2. The schema of the phases in physics.

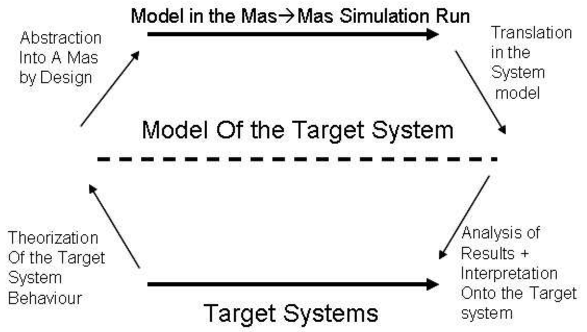


Fig. 3. Edmonds' schema revisited by the means of the introduction of another intermediate level.

perspective is not contrary either to what says Huges that in [7] makes clear that physical theories are not statements about physical world, on the contrary, they are statements about *theoretical constructs* and only if the theory is satisfactory then these constructs stand in a particular relation to the world. In next section some considerations about MABS and models are introduced.

The role of the mathematical model in physics is clear, and it appears to us in tune with the considerations about models that we saw in the previous paragraph.

III. Models and Phases of a Simulation

Our interpretation is that Multi-Agent based Simulations match many of the definitions of model that have been exposed in the precedent paragraphs. Edmonds in [6] states that MABS attempt to model Multi-Actor Systems with a Multi-Agent System. The attempt is to investigate a system by the construction of a MAS model and the analysis of the behavior of the MAS when it runs. In the opinion of Edmonds what distinguishes MABS from other forms of modeling is that simulations based on a Multi-Agent System adopt a MAS as formal model (in physics that role is of mathematics). Often in MABS object-actors or other entities of the system under study (the target system) are mapped onto agents in the MAS. Edmonds explains that the entities in the real system correspond to those of the agents in the MAS, while interactions between entities are correspondent to interactions rules between agents. Edmonds considers a MABS as a model of the target system. In his work he does remain at a high level and suggests interesting epistemological considerations about the MABS. On the basis of his work we would like to attempt a step further and try to detail the abstractions that constitute a MABS. A first consideration is that in a MABS reality is not directly mapped in a MAS, as one might be induced to think by looking casually at Edmonds' diagram, but what that is mapped it is a model of reality (see Figure 3). This model is an intermediate step between reality and MAS. Looking deeper at the different passages implied in the construction of a simulation model using a MAS, it appears clear that implicitly we are working also with other intermediate models. Each model represents a level of abstraction. In fact when we want to study a phenomenon by a computer simulation a first theoretical representation has to be translated through many others steps before it can run on a computer. For this reason to use computer simulation to do science, it must be kept a trace of all the passages that from reality bring to the software code.

Our proposal is schematized in Figure 4. In the left side of the schema are shown the existent levels that we identify between reality and software code. The circularity of the process of simulation described by Edmonds is maintained in our proposal. In the right side of the schema in Figure 4, in fact, are described the necessary steps that have to be followed to turn back simulation results to reality. In the next part of the paper we describe in details the meaning and the importance of each level shown in the schema. Edmonds in his article identifies also some phases in the simulation process (*Design*, *Inference*, *Analysis* and *Interpretation*) and we will see how they can be mapped in our schema at the end of next section. Now we will describe in details the meaning and the importance of each level shown in the schema.

A. Levels of Abstraction in simulation Building

In this section we give a brief explanation of each of the levels shown in the left side of our schema. The levels described below have to be considered levels of abstraction implied (although sometimes implicitly) in MABS. It is not in the scope of this work to give engineering guidelines to translate the requirements of a system into the software implementation, but we want to give a conceptual map of the structure of the practice of simulation.

1) **Target System**: this level of abstraction is constituted by the object of study that is determined by a specific point of view on a portion of reality, considered "isolated" from the rest of the universe. This first abstraction is the result of the identification of the problem that we want to examine and it is achieved by the Phase 1 (*Observation* in our schema). For example, the target system could be the urban road system if the goal of the simulation is to study the urban traffic problem. An example of a Multi-Agent based traffic model can be found in [14].
2) **Abstract Model**: the second level is the abstract model of the target system. The model construction (Phase 2) consists in the definition of the elements that are

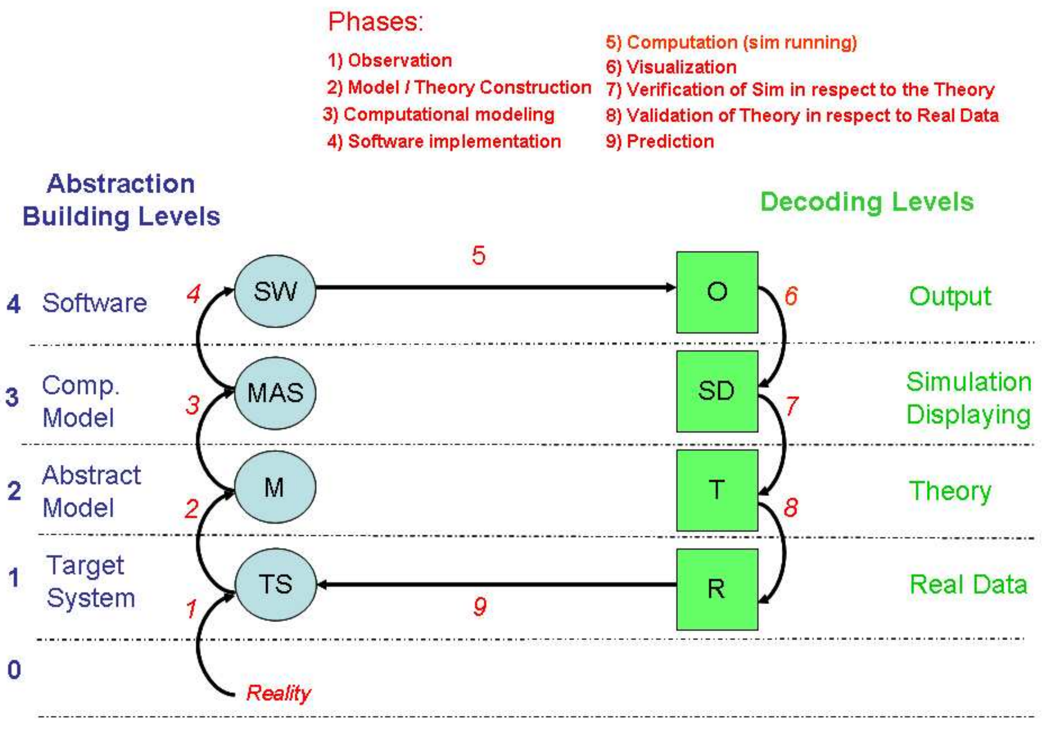

Phases:
1) Observation
2) Model / Theory Construction
3) Computational modeling
4) Software implementation
5) Computation (sim running)
6) Visualization
7) Verification of Sim in respect to the Theory
8) Validation of Theory in respect to Real Data
9) Prediction
Abstraction Building Levels
Decoding Levels
4 Software
3 Comp. Model
2 Abstract Model
1 Target System
0
SW
MAS
M
TS
Reality
O
SD
T
R
Output
Simulation Displaying
Theory
Real Data
1
2
3
4
5
6
7
8
9

Fig. 4. The schema shows the levels and phases involved in a simulation process. Notice that, with except of reality that has to be considered extra-model, the more abstract levels are found in the bottom part of the diagram.

considered constitutive of the Target System, and in the formulation of a set of hypothesis (conjectures that constitute our intuitive theory) about which rules govern their behaviors and interactions. This definition, at the beginning, can be also not rigorous. For example, in the case of the traffic, our abstract model is constituted by the elements of the environment (i.e. roads, crossings, traffic lights etc.), the individuals that populates the environment (cars, motorcycles etc.), the rules of behavior in this context (vehicles go along the road in one direction, they stop when the traffic light is red, they keep a security distance, etc.) and the properties of the elements that have been identified in that context (each vehicle has a speed etc.).

3) **Computational Model**: the third level is the specific computational model that has been adopted to represent the abstract model (see Phase 3 in the schema). The computational model is always a formal model (it can be a specific MAS). At this level the elements and the rules described in the precedent level are formally defined using proprieties and concepts of the MAS model (i.e. agents, signals etc). In the traffic example cars and traffic lights can be represented, in a MAS, by different kinds of agents, while the behavior of the single agents is modeled giving to the agents the possibility to emit and interpret specific signals (the red stop lights can be represented by a signal sent to car that is following).

4) **The software model**: the fourth level is the software translation of the Computational Model (Phase 4). The computational model, as it is, is an abstract definition and must be translated in a specific program language.

At the end of this 4 steps the original target system will be completely translated in something that is ready to be computed by a machine. The translation of that model into another implies the “encoding” of a model in the language of another. This is an activity that is intrinsic to the MABS and it is strictly related both to scientific activity and to technological constrains. It is important to be conscious of all these passages because each translation can introduce errors and sometimes also a lack of information due to the necessity to use a less expressive language (e.g. the description of the abstract model in the language of the MAS model). Therefore, each abstraction represents another step far from reality because it implies a transformation of the first model into something else. For this reason a “conversion key” that establishes what all the elements of the various models represents, is needed in order to maintain a correspondence with reality and use simulation for scientific investigation.

B. *Levels of Decoding*

The simulation process proceeds back in the direction of reality by steps that go through several abstraction levels and that help to check the effects of the assumptions of the previous phases and eventually to detect errors or limits of the adopted

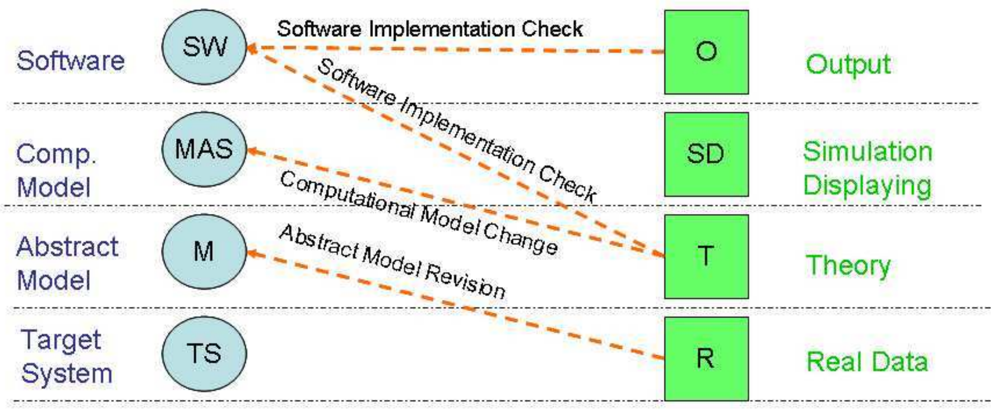


Fig. 5. This schema shows the levels involved in revision phases. of a simulation.

frameworks (e.g the selected MAS). Where it is necessary, in fact, may be necessary to come back to the previous levels and revise one or more of the models involved in the simulation. This operation allows also to correct mistakes introduced in translation phases, and to modify some models by the introduction of new elements that have been demonstrated to be necessary in order to obtain meaningful results (see Figure 5). The four decoding steps are presented below:

1) **Output**: outputs are the simple results of the software execution (Phase 5). If some errors are detected in this step it is necessary go back to the software code.
2) **Simulation Displaying**: the envisioning of the outputs (Phase 6) is often the only way to operate with the simulation dynamic data, because simple outputs are usable only at a machine level. This step is not trivial because consists in the representation of real entities by objects that do not necessarily offer a realistic visualization of the phenomena. The aim of Simulation Displaying in fact is to give an immediate and readable interpretation of the Outputs. This passage is often another abstraction jump that needs a translation key. The importance of the visualization of simulation output data has been largely discussed in many articles (i.e. see Batty and Smith in [15]). This step is very important, in particular for MAS-based simulations of complex systems, because the direct observation of the dynamics envisioned, allows to detect unexpected behaviors. These anomalies will be resolved in the next steps, and they could induce to a revision of the software, of the MAS, or the of Abstract Model. If we focus on the MAS-based traffic model example, at this level we can observe on the screen the dynamic evolution of the simulation constituted by the virtual cars that go along the roads.
3) **Theory**: after the visualization, the correctness of the simulation displaying data must “verified” (Phase 7); in other words it is necessary to check if our initial assumptions at the abstract level are captured by the simulation. This phase is not aimed at evaluating if the assumptions of the theory are correct, but only at verifying if these assumptions are maintained in the simulation. During this phase, we may need to go back to the software or to the MAS to correct mistakes and revise our computational modeling choices if we have some bad feedbacks from the displaying of the simulation. To turn back to traffic example, thanks to visualization phase we can observe the cars that move, brake, form a queue and so on. For example we could notice an anomalous behavior near crosses because cars do not give way correctly as we have assumed in our theory. Therefore we must go back to software in order to check if we did mistakes in the implementation or we may be forced to check if we have wrongly mapped the rules of the Abstract Model in the language of the computational model.
4) **Real Data**: after the verification phase next step is validation phase (Phase 8). If in the previous step we have decided that our theory is well represented by the simulation, now we can validate the theory in relation to real data collected in the target system. In this phase the results of the simulation must be related to real and measurable aspects of reality. In a study on the traffic problem, for example, a portion of a real street must be simulated and the verified results must be compared to the real data collected by observations of reality. If after this check simulation results are considered not reliable, it is necessary to turn back to the Abstract Model and change some of the initial assumptions, or it may be decided the introduction of new elements that at first time were wrongly considered not relevant for the representation of the Target System. The not realistic formation of very long queues, for example, could be resolved with a revision of the interaction rules between the cars and by the introduction of the possibility to overtake when the road on the other way is free.

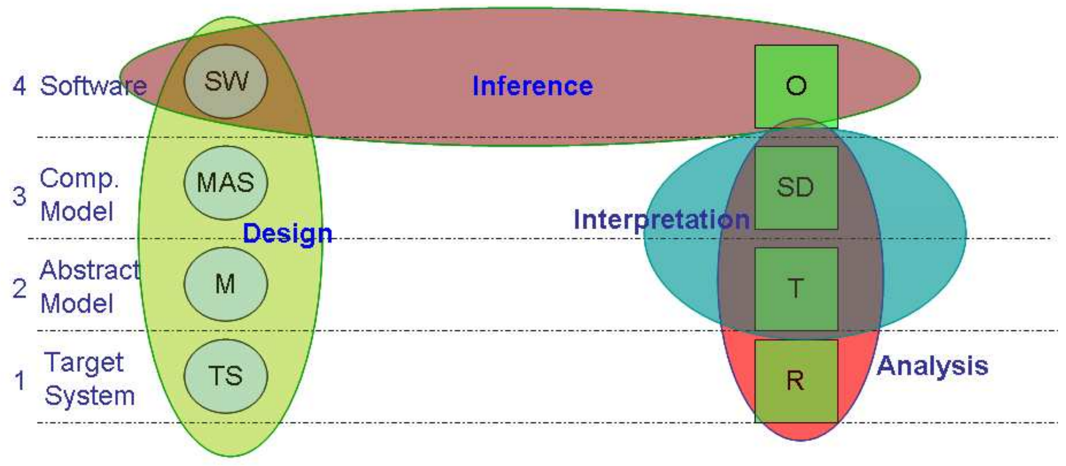


Fig. 6. The figure shows how classical phases of simulation process (as can be found in Edmonds' work) are mapped onto our schema

If the theory is validated by available data then it is possible to make predictions for future states of the Target System (Phase 9).

In reference to the work of Edmonds [6] the phases of the simulation process are identified as *Design*, *Inference*, *Analysis* and *Interpretation*. These phases in our schema are visible in Figure 6. The phase of Design in our schema involves all the 4 abstractions, as to say the Target System, the Abstract Model, the Computational Model and the Software. The Inference phase of Edmonds belongs to the level of the Software abstraction of our schema, the phase of Analysis involves from the level of Displaying to Real Data, while Interpretation regards Simulation Displaying and the Theory.

IV. Some results for the simulation practice

In this section the introduction of an example will clarify our considerations about the aid that the conceptualization given in the proposed schema can give to simulation activity. Therefore we will now introduce the work described in [16]. In particular Balmer and Nagel describe a MAS simulation focused on traffic roundabouts which is applied to a specific Zurich area. During the phase of Design the authors decided to represent cars as particles that move along tunnels (driving routes) representing streets. The spatial model is continuous (the position of each particle is determined by a pair of coordinates) and the particles can be considered agents because they have specific behavior rule (agents must respect the physical rules of acceleration, they have a specific desired speed, they must respect other agents in the system decelerating or overtaking if a slower agent drive in front of them, etc.). Agents also hold information about their destination. Therefore the target system is also in this case the traffic dynamics, the abstract model is constituted by the assumptions and rules like the ones presented above, while the chosen computational model is constituted by a set of physical equations describing the particles motion. During what we have called "the analysis", in particular in the phase that in our schema is identified with the number 8 (validation), a problem was detected: in some situation the cars were subjected to forces exerted in opposite direction (the forces implied were the one directed towards the desired way and the repulsion force aimed at the avoidance of collisions with other cars) and they became unable to move. This problem could not be identified in the design process because it can be considered as an emergent phenomenon that occurs only in some situations and in some specific spatial contexts. This event is a relevant simulation result, because it can be a hint of a possible congestion. In this framework it is then useful to record it, but since this is not a routinary event, the behavior of agent drivers in this situation cannot modelled in a simple way. The decision in this case could have been to turn back to the abstract model and operate a change (i.e. discretization of space or the introduction of giving way rules). However Balmer and Nigel anyway decided not to operate these big changes of their abstract model, but they opted for a minor modification: the introduction of a compenetration rule in order to force the cars to go beyond the stall situation and avoid the deadlock. This is an abstract and gross representation of a set of complex behaviors that can be assumed by human drivers to solve a stall. Thanks to this solution the global behavior of the cars (in sense of aggregate data) maintains a better similarity with the observed reality. In our opinion Balmer and Nigel could detect the problem and adopt a good solution for a correct and scientific use of simulation because they were aware of all the steps done in the previous design phase. It is fundamental to be aware of all these passages as it would be an error to map directly the real world in the software code. if we follow strictly all the passages shown in the schema, it is possible to keep track of the translations of our initial assumptions and, in this way, we can easily avoid the introduction of arbitrary details whose influence cannot be kept under control, as pointed out by some of the critics to simulation practice that we summarized in Section I.

We then suggested that simulations can be used as a valid tools for scientific investigation as the cycle of a simulation

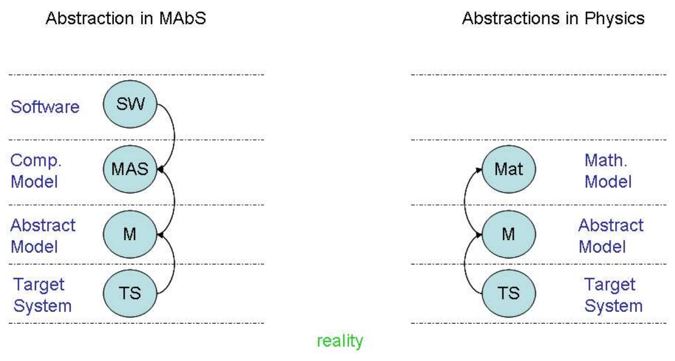


Fig. 7. The schema in figure shows on the left side the abstractions level implied in a simulation in comparison to those (on the right side) implied by experimental science.

reflects the cycle of a science like physics, although a further abstraction is involved in simulation as it is shown in Figure 7. If we come back to reality following the proposed schema in all its passages it is possible to obtain meaningful results that are connected with reality, especially if the revision phases are performed accurately. This practice is analogous to the one followed by a science like physics where a first phase of simplification and abstraction (in newtonian physics reality is so simplified that bodies are seen just as simple points without extension, see [12]) follows an experimental phase by which the results obtained in the simplified model are projected back to reality. These considerations can help to give an answer to other of the objections presented at the beginning of the paper.

V. Conclusions

In this paper we began reporting some definitions of what can be considered a model in science and we briefly described the scientific cycle in physics. Then we raised some questions about MABS and their role in scientific investigation. We started from the work of Edmonds to introduce our investigation in the nature of simulations in the attempt to give some answers. Our conclusion is that simulations are not “models” but are a compound of many models at different levels of abstractions. We pointed out that in the encoding or decoding of one level into another is situated the risk to loose the scientific purpose of simulations, if a translation criteria between models is not stated explicitly. When one modeler is led to revise his/her theory about the simulated domain after an analysis of simulation results and their validation, simulation can be considered a useful tool for scientific investigation. On the other hand, if the simulation has been deeply tested and we can trust simulation results, we focus on the phase of simulation prediction. In this last case we are working with a tested instrument and we are not using MABS to do scientific investigation but simply, for example, to help decision makers. It must be noted that other areas of Computer Science and engineering deal with models and abstractions of real systems, in particular Software Engineering. For example the problem of correspondence between models is considered also by the Model Driven Architecture Approach (MDA), that focuses on the importance of the mapping between the models present at different levels of Software Engineering practice. One of the purpose of MDA is to give guidelines for integration of Information Technologies in order to assure interoperability between systems (for an introduction and first references on MDA see for example [17]). Another meaningful example is the work of M. Jackson [18] that analyzes under the perspective of Problem Frames (PF) the relation that, in Software Engineering, exists between the requirements, the real world, and the machine considered as the general purpose computer that will execute the software. MABS share several aspects with Software Engineering, because the proposed schema reminds several iterative software development processes that are used also to project and build an effective simulation tool. MDA or PF definitions take in consideration a class of problems that is involved also in MABS, but our study does not take into consideration deeply the technical problems that are strictly related to the machine, the software and the specific available technologies. The aim of this work is in fact to state some general considerations that could help to preserve scientific contents through the many models involved in MAS simulation practice.

References

[1] E. Winsberg, “Sanctioning models: epistemology of simulation”, Science in Context, 12, 275-292, 1999.
[2] D.C. Dennett, “Real Patterns”, The Journal of Philosophy, Vol. 88, No. 01, pp. 27-51, 1991.
[3] K. Richardson, “The Problematisation of Existence: Towards a Philosophy of Complexity”, In proceedings of ISCE Workshop on Complexity and Philosophy, 2002.
[4] D. Parisi, “Simulazioni”, Il Mulino Eds, Bologna, 2001.
[5] J. Doran and N. Gilbert, “Simulating Societies: an introduction”, in N.Gilbert e J.Doran Eds, Simulating Societies: the Computer Simulation of Social Phenomena, UCL Press, pp. 1-18, London, 1994.
[6] B. Edmonds, “The Use of Models - Making MABS More Informative”, Lecture Notes in Computer Science, Page 15, 1979, Jan 2000, .
[7] R.I.G. Hughes, “Models and Representation”, Philosophy of Science, pp. S325-S336, 1997.
[8] E. Winsberg, “Simulated Experiments: Methodology for a virtual World”, Philosophy of Science, pp. 105-125, 2003.
[9] M. Morrison and M.S. Morgan, “Models as Mediators, Perspectives on Natural and Social Science”, Cambridge University Press, 1999.
[10] N. Cartright in “the dappled world. A study of the boundaries of Science”, Cambridge, Cambridge UP 1999.
[11] M.L. Dalla Chiara and G. Toraldo di Francia, “Introduzione alla filosofia della scienza”, Laterza, Roma-Bari, 1999.
[12] D. Hestenes, “Modeling Games in the Newtonian World”, Am. J. Phys. pp. 732-748, (1992).
[13] A. Gelfert, “Simulating Many-Body Models in Physics: Rigorous Results, ’Benchmarks’, and Cross-Model Justification”, in Proceedings Models and Simulations, London, 2006.
[14] K. Dresner and P. Stone, “Multiagent Traffic Management: A Reservation-Based Intersection Control Mechanism”, In Proceedings of the Third International Joint Conference on Autonomous Agents and MultiAgent Systems, p. 530–537, 2004.
[15] M. Batty and A. Smith, “Urban Simulacra: From Real to Virtual Cities and Back and Beyond”, Architectural Design, Sensing the 21st Century City: The Net City Close-up and Remote, Eds. David Grahame Shane and Brian McGrath, 2005.
[16] M. Balmer and K. Nagel, “Shape Morphing of Intersection Layout Using Curb Side Oriented Driver Simulation”, in Innovations in Design & Decision Support Systems in Architecture and Urban Planning, Springer–Verlag, p. 167–183, 2006.
[17] A. Wegmann and O. Preiss, “Strengthening MDA by Drawing from the Living System Theory”, in WiSME@UML 2002, Workshop in Software Model Engineering, Dresden, Germany, 2002.
[18] M. Jackson, “Problem Frames and Software Engineering”, Information and Software Technology 47, p. 903–912, 2005.

Conformance and Interoperability in Open Enviroments

Matteo Baldoni, Cristina Baroglio,
Alberto Martelli, and Viviana Patti
Dipartimento di Informatica — Università degli Studi di Torino
C.so Svizzera, 185 — I-10149 Torino (Italy)
Email: {baldoni,baroglio,mrt,patti}@di.unito.it

Abstract— **An important issue, in open environments like the web, is guaranteeing the interoperability of a set of services. When the interaction scheme that the services should follow is given (e.g. as a choreography or as an interaction protocol), it becomes possible to verify, before the interaction takes place, if the interactive behavior of a service (e.g. a BPEL process specification) respects it. This verification is known as "conformance test". Recently some attempts have been done for defining conformance tests w.r.t. a protocol but these approaches fail in capturing the very nature of interoperability, turning out to be too restrictive. In this work we give a representation of protocol, based on message exchange and on finite state automata, and we focus on those properties that are essential to the verification the interoperability of a set of services. In particular, we define a conformance test that can guarantee, a priori, the interoperability of a set of services by verifying properties of the single service against the protocol. This is particularly relevant in open environments, where services are identified and composed on demand and dynamically, and the system as a whole cannot be analyzed.**

I. Introduction

In this work we face the problem of verifying the interoperability of a set of peers by exploiting an abstract description of the desired interaction. On the one hand, we will have an interaction protocol (possibly expressed by a choreography), capturing the global interaction of a desired system of services; on the other, we will have a set of service implementations which should be used to assemble the system. The protocol is a specification of the desired interaction, as thus, it might be used for defining several systems of services [3]. In particular, it contains a characterization of the various *roles* played by the services [6]. In our view, a role specification is not the exact specification of a process of interest, rather it identifies a *set of possible processes*, all those whose evolutions respect the dictates given by the role. In an open environment, the introduction of a new peer in an execution context will be determined provided that it satisfies the protocol that characterizes such an execution context; as long as the new entity satisfies the rules, the interoperability with the other components of the system is guaranteed.

The computational model to which web services are inspired is that of *distributed objects* [10]. An object cannot refuse to execute a method which is invoked on it and that is contained in its public interface, in the very same way as a service cannot refuse to execute over an invocation that respects its public interface (although it can refuse the answer). This, however, is not the only possible model of execution. In multi-agent systems, for instance, an agent sending a request message to another agent cannot be certain that it will ever be answered, unless the interaction is ruled by a protocol. The protocol plays, in a way, the role of the public interface: an agent conforming to a protocol must necessarily answer and must be able to handle messages sent by other agents in the context of the protocol itself. The difference between the case of objects and the case of protocols is that the protocol also defines an "execution context" in which using messages. Therefore, the set of messages that it is possible to use varies depending on the point at which the execution has arrived. In a way, the protocol is a *dynamic interface* that defines messages in the context of the occurring interaction, thus ruling this interaction. On the other hand, the user of an object is not obliged to use all of the methods offered in the public interface and it can implement more methods. The same holds when protocols are used to norm the interaction. Generally speaking, only part of the protocol will be used in an entity's interaction with another and they can understand more messages than the one forseen by the protocol. Moreover, we will assume that the initiative is taken from the entity that plays as a sender, which will commit to sending a specific message out of its set of alternatives. The receiver will simply execute the reception of the message. Of course, the senders should send a message that its counterpart can understand. For all these reasons, performing the conformance test is analogous to verifying at compilation time (that is, a priori) if a class implements an interface in a correct way and to execute a static typechecking.

Sticking to a specification, on the other hand, does not mean that the service must do *all* that the role specification defines; indeed, a role specification is just a formal definition of what is lawful to say or to expect at any given moment of the interaction. Taking this observation into account we need to define some means for verifying that a single service implementation comforms to the specification of the role in the protocol that it means to play [14]. The idea is that if a service passes the conformance test it will be able to interact with a set of other services, equally proved individually conformant to the other roles in the protocol, in a way that respects the rules defined in the protocol itself.

A typical approach to the verification that a service implementation respects a role definition is to verify whether the execution traces of the service belong to the protocol [1], [12], [7]. This test, however, does not consider processes with different branching structures. Another approach, that instead takes this case into account, is to apply bisimulation and say that the implementation is conformant if it is bisimilar to its role or, more generally, that the composition of a set of policies is bisimilar to the composition of a set of roles [9], [18]. Bisimulation [16], however, does not take into account the fact that the implementor's decisions of cutting some interaction path not necessarily compromise the interaction. Many services that respect the intuitions given above will not be bisimilar to the specification. Nevertheless, it would be very restrictive to say that they are not conformant (see Section III-A). Thus, in order to perform the conformance test we need a softer test, a test that accepts all the processes contained in a space defined by the role. In this work we provide such a test (Section III). This proposal differs from previous work that we have done on conformance [7], [8] in various aspects. First of all, we can now tackle protocols that contain an arbitrary (though finite) number of roles. Second, we account also for the case of policies and roles which produce the same interactions but have different branching structures. This case could not be handled in the previous framework due to the fact that we based it exclusively on a trace semantics.

II. Protocols, policies, and conversations

A *conversation policy* is a program that defines the communicative behavior of an interactive entity, e.g. a service, implemented in some programming language [3]. A conversation *protocol* specifies the desired communicative behavior of a set of interactive entities. More specifically, a conversation protocol specifies the sequences of messages (also called speech acts) that can possibly be exchanged by the involved parties, and that we consider as lawful.

In languages that account for communication, speech acts often have the form $m(a_s,\ a_r,\ l)$, where m is the kind of message, or performative, a_s (sender) and a_r (receiver) are two interactive entities and l is the message content. In the following analysis it is important to distinguish the incoming messages from the outgoing messages w.r.t a role of a protocol or a policy. We will write m? (*incoming message*) and m! (*outgoing message*) when the receiver or the utterer and the content of the message is clear from the context or they are not relevant. So, for instance, $m(a_s,\ a_r,\ l)$ is written as m? from the point of view of a_r, and m! from the point of view of the sender. By the term *conversation* we will, then, denote a sequence of speech acts that is a dialogue of a set of parties.

Both a protocol and a policy can be seen as sets of conversations. In the case of the protocol, it is intuitive that it will be the set of all the possible conversations allowed by its specification among the partners. In the case of the single policy, it will be the set of the possible conversations that the entity can carry on according to its implementing program. Although at execution time, depending on the interlocutor and on the circumstances, only one conversation at a time will actually be expressed, in order to verify conformance *a priori* we need to consider them all as a set. It is important to remark before proceeding that other proposal, e.g. [2], focus on a different kind of conformance: *run-time* conformance, in which only the ongoing conversation is checked against a protocol.

Let us then introduce a formal representation of policies and protocols. We will use *finite state automata* (FSA). This choice, though simple, is the same used by the well-known verification system SPIN [15], whose notation we adopt. FSA will be used for representing individual processes that exchange messages with other processes. Therefore, FSA will be used both for representing the *roles* of a protocol, i.e. the abstract descriptions of the interacting parties, as well as for representing the policies of specific entities involved in the interaction. In this work we do not consider the translation process necessary to turn a protocol (e.g. a WS-CDL choreography) or an entity's policy (e.g. a BPEL process) in a FSA; our focus is, in fact, conformance and interoperability. It is possible to find in the literature some works that do this kind of translations. An example is [12].

Definition 2.1 (Finite State Automaton): A finite state automaton is a tuple (S, s_0, L, T, F), where S is a finite set of states, $s_0 \in S$ is a distinguished initial state, L is a finite set of labels, $T \subseteq (S \times L \times S)$ is a set of transitions, $F \in S$ is a set of final states.
Similarly to [15] we will denote by the "dot" notation the components of a FSA, for example we use $A.s$ to denote the state s that belongs to the automaton A. The definition of run is taken from [15].

Definition 2.2 (Runs and strings): A run σ of a FSA (S, s_0, L, T, F) is an ordered, possibly infinite, set of transitions (a sequence) $(s_0, l_0, s_1), (s_1, l_1, s_2), (s_2, l_2, s_3), \ldots$ such that $\forall i \geq 0$, $(s_i, l_i, s_{i+1}) \in T$, while the sequence $l_0 l_1 \ldots$ is the corresponding string $\overline{\sigma}$.

Definition 2.3 (Acceptance): An accepting run of a finite state automaton (S, s_0, L, T, F) is a finite run σ in which the final transition (s_{n-1}, l_{n-1}, s_n) has the property that $s_n \in F$. The corresponding string $\overline{\sigma}$ is an accepted string.

Given a FSA A, we say that a state $A.s_1 \in A.S$ is *alive* if there exists a finite run $(s_1, l_1, s_2), \ldots, (s_{n-1}, l_{n-1}, s_n)$ and $s_n \in A.F$. Moreover, we will write $A_1 \subseteq A_2$ iff every string of A_1 is also a string of A_2.

In order to represent compositions of policies or of individual protocol roles we need to introduce the notions of *free* and of *synchronous* product. These definitions are an adaptation to the problem that we are tackling of the analogous ones presented in [4] for Finite Transition Systems.

Definition 2.4 (Free product): Let A_i, $i = 1, \ldots, n$, be n FSA's. The *free product* $A_1 \times \cdots \times A_n$ is the FSA $A = (S, s_0, L, T, F)$ defined by:

- S is the set $A_1.S \times \cdots \times A_n.S$;
- s_0 is the tuple $(A_1.s_0, \ldots, A_n.s_0)$;
- L is the set $A_1.L \times \cdots \times A_n.L$;

- T is the set of tuples $((A_1.s_1, \ldots, A_n.s_n), (l_1, \ldots, l_n), (A_1.s'_1, \ldots, A_n.s'_n))$ such that $(A_i.s_i, l_i, A_i.s'_i) \in A_i.T$, for $i = 1, \ldots, n$; and
- F is the set of tuples $(A_1.s_1, \ldots, A_n.s_n) \in A.S$ such that $s_i \in A_i.F$, for $i = 1, \ldots, n$.

We will assume, from now on, that every FSA A has an *empty* transition (s, ε, s) for every state $s \in A.S$. When the finite set of labels L used in a FSA is a set of *speech acts*, strings will represent *conversations*.

Definition 2.5 (Synchronous product): Let A_i, $i = 1, \ldots, n$, be n FSA's. The *synchronous product* of the A_i's, written $A_1 \otimes \cdots \otimes A_n$, is the FSA obtained as the free product of the A_i's containing only the transitions $((A_1.s_1, \ldots, A_n.s_n), (l_1, \ldots, l_n), (A_1.s'_1, \ldots, A_n.s'_n))$ such that there exist i and j, $1 \leq i \neq j \leq n$, $l_i = \mathsf{m}!$, $l_j = \mathsf{m}?$, and for any k not equal to i and j, $l_k = \varepsilon$.

The synchronous product allows a system that exchanges messages to be represented. It is worth noting that a synchronous product does not imply that messages will be exchanged in a synchronous way; it simply represents a message exchange without any assumption on how the exchange is carried on.

In order to represent a protocol, we use the synchronous product of the set of such FSA's associated with each role (where each FSA represents the communicative behavior of the role). Moreover, we will assume that the automata that compound the synchronous product have some "good properties", which meet the commonly shared intuitions behind protocols. In particular, we assume that for the set of such automata the following properties hold:

1) *any message that can possibly be sent, at any point of the execution, will be handled by one of its interlocutor*;
2) *whatever point of conversation has been reached, there is a way to bring it to an end*.

An arbitrary synchronous product of n FSA's might not meet these requirements, which can, however, be verified by using automated systems, like SPIN [15].

Note that protocol specification languages, like UML sequence (activity) diagrams and automata [17], naturally follow these requirements: an arrow starts from the lifeline of a role, ending into the lifeline of another role, and thus corresponds to an outgoing or to an incoming message depending on the point of view. Making an analogy with the computational model of distributed objects, one could say that the only messages that are sent are those which can be understood. Moreover, usually protocols contain finite conversations.

We will say that a conversation is *legal w.r.t. a protocol* if it respects the specifications given by the protocol, i.e. if it is an accepted string of the protocol.

III. Interoperability and Conformance Test

We are now in position to explain, with the help of a few simple examples, the intuition behind the terms "conformance" and "interoperability", that we will, then, formalize. By *interoperability* we mean the capability of a set of entities of actually producing a conversation when interacting with one another [5]. Interoperability is a *desired property* of a system of interactive entities and its verification is fundamental in order to understand whether the system works. Such a test passes through the analysis of all the entities involved in the interaction.

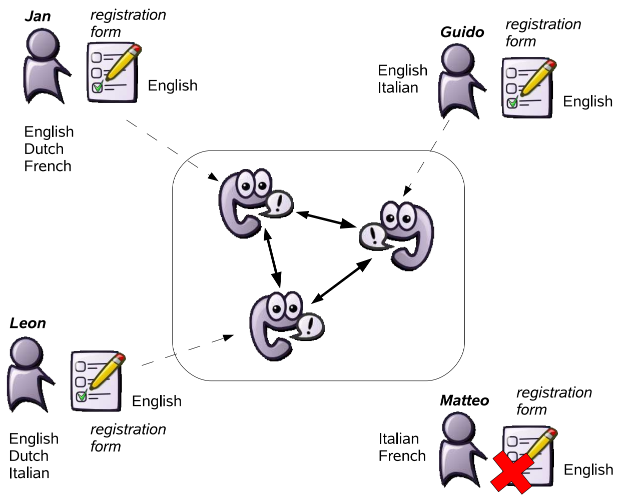


Fig. 1. Example of the summer school.

Figure 1 shows an intuitive example, in which a group of persons wish to attend a summer school. Each of them can speak and understand different languages. For instance, Jan can speak English, Dutch, and French. The school registration form requires the interested attendee to speak and understand English, which is the official language of the school. This requirement allows a person to decide if it will be in condition to interact with the other participants before attending the school. So, for instance, Matteo, who speaks Italian and could therefore interact with Guido, will not be in condition to understand the other participants. Jan and Leon could interact by speaking Ducth, however, since they also know English, they will be able to interact with all the other attendees and so they will be in condition to participate. The fact that they understand other languages besides the one required by the "school protocol" does not compromise their interoperability with the others. In fact, within the context of the school everybody will speak English with them. Interoperability is compromised when one does not understand (part of) the protocol (e.g. Matteo) or when one decides to speak a language that is not agreed (e.g. Leon when speaking Dutch).

In an open system, however, it is quite unlikely to have a global view of the system either because it is not possible to read part of the necessary information (e.g. some services do not publish their behavior) or because the interactive entities are identified at different moments, when necessary. Protocols are adopted to solve such problems, in fact, having an interaction schema allows the *distribution of the tests in time*, by checking a single entity at a time against the role that it should play. The protocol, by its own nature guarantees the interoperability of the roles that are part of it. One might argue why we do not simply verify the system obtained by substituting the policy instead of its role within the protocol and, then, check whether any message that can be sent will be handled by some of the interlocutor roles, bringing to an end the conversations. Actually, this solution presents some flaws, as the following counter-example proves. Let us consider a protocol with three roles: A_1 sends m_1 to A_2, A_2 waits for m_1 and then it waits for m_2, and A_3 sends m_2 to A_2. Let us know substitute to role A_2 the policy which, first, waits for m_2 and then it waits for m_1. The three partners will perfectly interoperate and successfully conclude their conversations but the conversation that is produced is not legal w.r.t. the protocol. In protocol-based systems, the proof of the interoperability of an entity with others, obtained by checking the communicative behavior of the entity against the rules of the system (i.e. against an *interaction protocol* itself), is known as *conformance test*. Intuitively, this test must guarantee the following *definition of interoperability*.

Definition 3.1 (Interoperability w.r.t. an interaction protocol): Interoperability w.r.t. an interaction protocol is the capability of a set of entities of producing a conversation that is legal w.r.t. the protocol.

Let us now consider a given service that should play a role in a protocol. In order to include it in the interaction we need to understand if it will be able to interact with the possible players of the other roles. If we assume that the other players are conformant to their respective roles, we can represent them by the roles themselves. Roles, by the definition of protocol, are interoperable. Therefore, in order to prove the interoperability of our service, it will be sufficient to prove for it the "good properties" of its role. First of all, we should prove that its policy does not send messages that the others cannot understand, which means that it will not send messages that are not accounted for by the role. Moreover, we should prove that it can tackle every incoming message that the other roles might send to it, which means that it must be able to handle all the incoming messages handled by the role. Another important property is that whatever point of conversation has been reached, there is a way to bring it to an end. In practice, if a role can bring to an end a conversation in which it has been engaged, so must do the service. To summarize, in order to check a service interoperability it will be sufficient to check its *conformance w.r.t. the desired role* and this check will guarantee that the service will be able to interact with services equally, and separately, proved conformant to the other roles. This, nevertheless, does not mean that the policy of the service must be a precise "copy" of the role.

A. Expectations for interoperability

Let us now discuss some typical cases in which a policy and a role specification that differ in various ways are compared in order to decide if the policy conforms to the role so as to guarantee its interoperability with its future interlocutors that will play the other roles in the protocol. With reference to Figure 2, let us begin with considering the case reported in row (a): here, the service can possibly utter a message m_3 that is not foreseen by the role specification. Trivially,

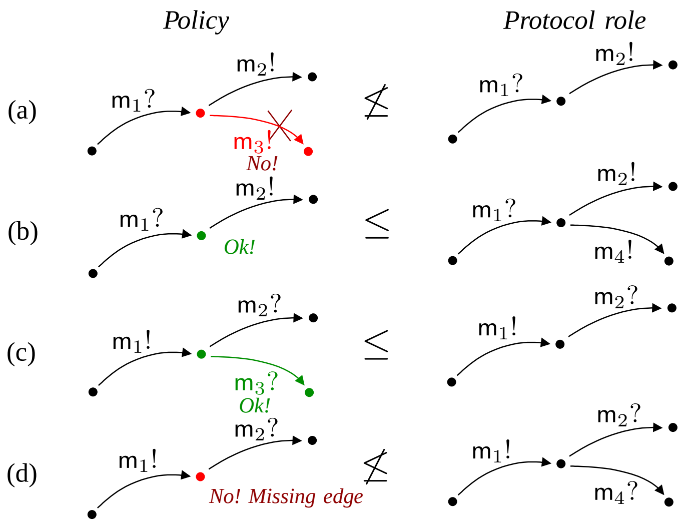


Fig. 2. A set of cases that exemplifies our expectations about a conformant policy: cases (b) and (c) do not compromise interoperability, hence they should pass the conformance test; cases (a) and (d) instead should not pass the conformance test.

this policy is *not conformant* to the protocol because the service might send a message that cannot be handled by any interlocutor that conforms to the protocol. The symmetric case in which the policy accounts for less outgoing messages than the role specification (Figure 2, row (b)) is, instead, legal. The reason is that at any point of its conversations the entity will anyway always utter only messages that the entities playing the other roles will surely understand. Hence, interoperability is preserved. The restriction of the set of possible alternatives (w.r.t. the protocol) depends on the implementor's own criteria.

Let us now consider the case reported in Figure 2, row (c). Here, the service policy accounts for two conversations in which, after uttering a message m_1, the entity expects one of the two messages m_2 or m_3. Let us also suppose that the protocol specification only allows the first conversation, i.e. that the only possible incoming message is m_2. When the entity will interact with another that is conformant to the protocol, the message m_3 will never be received because the other entity will never utter it. So, in this case, we would like the a priori conformance test to accept the policy as *conformant* to the specification.

Talking about incoming messages, let us now consider the symmetric case (Figure 2, row (d)), in which the *protocol specification* states that after an outgoing message m_1, an answer m_2 or m_4 will be received, while the policy accounts only for the incoming message m_2. In this case, the expectation is that the policy is *not conformant* because there is a possible incoming message (the one with answer m_4) that can be enacted by the *interlocutor*, which, however, cannot be handled by the policy. This compromises interoperability.

To summarize, at every point of a conversation, we expect that a conformant policy never utters speech acts that are not expected, according to the protocol, and we also expect it to be able to handle any message that can possibly be received, once

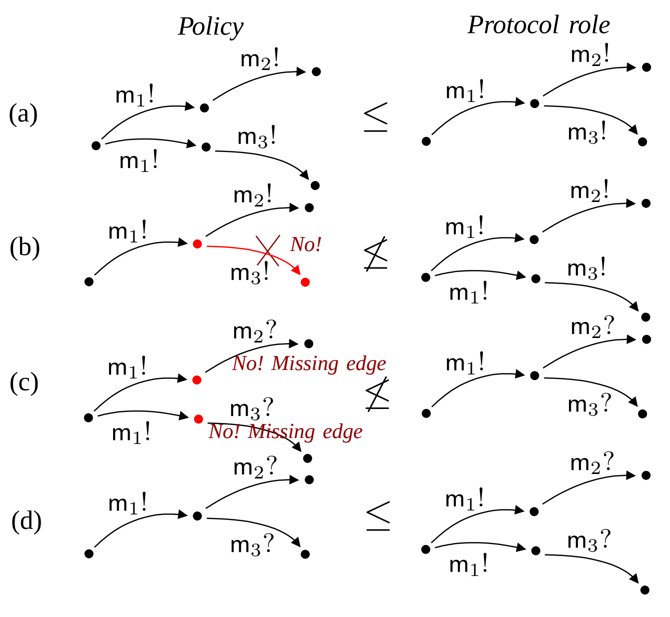


Fig. 3. A set of cases that exemplifies our expectations about a conformant policy: differently than in Figure 2, for every row, the policy and the role produce the same conversations but the structure of their implementations differ.

again according to the protocol. However, the policy is not obliged to foresee (at every point of conversation) an outgoing message for every alternative included in the protocol but it must foresee at least one of them if this is necessary to proceed with the conversation. Trivially, in the example of row (b), a policy containing only the conversation $\mathsf{m}_1?$ (not followed either by $\mathsf{m}_2!$ or by $\mathsf{m}_4!$) would not be *conformant*.

Let us now consider a completely different set of situations, in which the "structure" of the policy implemented and the structure of the role specification are taken into account. These situations are taken from the literature on communicating processes [13]. Figure 3 reports a set of cases in which the role description and the policy allow the *same conversations* but their structure differs: in rows (a) and (c) the policy decides which message to send (receive, respectively) after m_1 from the very beginning, while in the protocol this decision is taken after m_1 is sent. In row (b) and (d) the situation is inverted.

The case of row (a) does not compromise conformance in the same way as the case reported at row (b) of Figure 2 does not: after a non-deterministic choice the set of alternative outgoing messages is restricted but in both cases only legal messages that can be handled by the interlocutor will be sent. The analogous case reported in row (c), concerning incoming messages, instead, compromises the conformance. In fact, after the non-deterministic step the policy might receive a message that it cannot handle, similarly to row (d) of Figure 2.

The case of row (b), Figure 3, compromises the conformance because after the non-deterministic choice the role specification allows a single outgoing message with no alternatives. The policy, instead, might utter one out of two alternative messages (similarly to row (a) of Figure 2). Finally, the case of row (d) does not compromise the conformance, following what reported in Figure 2, row (c).

B. *Conformance and interoperability*

In this section we define a test, for checking conformance, that is derived from the observations above. A first consideration is that a conformance test *is not an inclusion test* w.r.t. the set of possible conversations that are produced. In fact, for instance, in row (d) of Figure 2 the policy produces a subset of the conversations produced by the role specification but interoperability is not guaranteed. Instead, if we consider row (c) in the same figure, the set of conversation traces, produced by the policy, is a superset of the one produced by the protocol; despite this, interoperability is guaranteed. A second consideration is that a conformance test is not a bisimulation test w.r.t. the role specification. Actually, the (bi)simulation-based test defined in concurrency theory [16] is too strict, and it imposes constraints, that would exclude policies which instead would be able to interoperate, within the context given by the protocol specification. In particular, all the cases reported in Figure 3 would not be considered as conformant because they are all pairs of processes with different branching structures. Despite this, we would like our test to recognize cases (a) and (d) as conformant because they do not compromise interoperability.

The solution that we propose is inspired by (bi)simulation, but it distinguishes the ways in which incoming and outgoing messages are handled, when a policy is compared to a role [1]. In the following, we will use "$A_1 \leq A_2$" to denote the fact that A_1 conforms to A_2. This choice might seem contradictory after the previous discussion, in fact, in general $A_1 \leq A_2$ does not entail $A_1 \subseteq A_2$. However, with symbol "$\leq$" we capture the fact that A_1 will actually produce a subset of the conversations forseen by the role, *when interacting with entities that play the other roles in the protocol* (see Propositions 3.3 and 3.4). This is what we expect from a conformant policy and from our definition of interoperability. Let A an FSA, let us denote by $\text{Succ}(l, s)$ the set of states $\{s' \mid (s, l, s') \in A.T\}$.

Definition 3.2 (Conformant simulation): Given two FSA's A_1 and A_2, A_1 is a *conformant simulation* of A_2, written $A_1 \leq A_2$ iff there is a binary relation $\mathcal{R}$ between A_1 and A_2 such that

1) $A_1.s_0 \mathcal{R} A_2.s_0$;
2) if $s_i \mathcal{R} s_j$, where $s_i \in A_1.S$ and $s_j \in A_2.S$, then
   a) for every $(s_i, \mathsf{m}!, s_{i+1}) \in A_1.T$, $\text{Succ}(\mathsf{m}!, s_j) \neq \emptyset$ and $s_{i+1} \mathcal{R} s'$ for every $s' \in \text{Succ}(\mathsf{m}!, s_j)$;
   b) for every $(s_j, \mathsf{m}?, s_{j+1}) \in A_2.T$, $\text{Succ}(\mathsf{m}?, s_i) \neq \emptyset$ and $s_{j+1} \mathcal{R} s'$ for every $s' \in \text{Succ}(\mathsf{m}?, s_i)$;

Particularly relevant is the case in which A_2 is a role in a protocol and A_1 is a policy implementation. Notice that, in this case, conformance is defined only w.r.t. the role that the single policy implements, *independently* from the rest of the protocol. As anticipated above, Definition 3.2 does not

[1] All proofs are omitted for lack of space, they will be supplied on demand.

imply the fact that "$A_1 \leq A_2$ entails $A_1 \subseteq A_2$". Instead, the following proposition holds.

Proposition 3.3: Let $A_1 \otimes \cdots \otimes A_i \otimes \cdots \otimes A_n$ be a protocol, and A'_i a policy such that $A'_i \leq A_i$, then $A_1 \otimes \cdots \otimes A'_i \otimes \cdots \otimes A_n \subseteq A_1 \otimes \cdots \otimes A_i \otimes \cdots \otimes A_n$.
This proposition catches the intuition that a conformant policy is able to produce a subset of the legal conversations defined by the protocol but only when it is executed in the context given by the protocol.

The above proposition can be generalized in the following way. Here we consider a set of policies that have been individually proved as being conformant simulations of the various roles in a protocol. The property states that the dialogues that such policies can produce will be legal *w.r.t. the protocol*.

Proposition 3.4: Let $A_1 \otimes \cdots \otimes A_n$ be a protocol and let $A'_1, \ldots, A'_n$ be n policies such that $A'_i \leq A_i$, for $i = 1, \ldots, n$, then $A'_1 \otimes \cdots \otimes A'_n \subseteq A_1 \otimes \cdots \otimes A_n$
In order to prove interoperability we need to prove that our policies will actually produce a conversation when interacting, while so far we have only proved that if a conversation will be generated, it will be legal. By assumption, in a protocol it is always possible to conclude a conversation whatever the point at which the interaction arrived. We expect a similar property to hold also for a set of policies that have been proved conformant to the roles of a protocol. The relation $\leq$ is too weak, so we need to introduce the notion of *complete conformant simulation*.

Definition 3.5 (Complete conformant simulation): Given two FSA's A_1 and A_2 we say that A_1 is a *complete conformant simulation* of A_2, written $A_1 \trianglelefteq A_2$, iff there is a A_1 is a conformant simulation of A_2 under a binary relation $\mathcal{R}$ and

- for all $s_i \in A_1.F$ such that $s_i \mathcal{R} s_j$, then $s_j \in A_2.F$;
- for all $s_j \in A_2.S$ such that s_j is alive and $s_i \mathcal{R} s_j$, $s_i \in A_1.S$, then s_i is alive.

Now, we are in the position to give the following fundamental result.

Theorem 3.6 (Interoperability): Let $A_1 \otimes \cdots \otimes A_n$ be a protocol and let $A'_1, \ldots, A'_n$ be n policies such that $A'_i \trianglelefteq A_i$, for $i = 1, \ldots, n$. For any common string $\overline{\sigma'}$ of $A'_1 \otimes \cdots \otimes A'_n$ and $A_1 \otimes \cdots \otimes A_n$ there is a run $\sigma'\sigma''$ such that $\overline{\sigma'\sigma''}$ is an accepted string of $A'_1 \otimes \cdots \otimes A'_n$.
Intuitively, whenever two policies, that have independently been proved conformant to the two roles of a protocol, start an interaction, thanks to Proposition 3.4, they will be able to conclude their interaction producing a legal accepted run. Therefore, Theorem 3.6 implies Definition 3.1 (interoperability).

IV. Conclusions and related works

In this work we have given a definition of conformance and of interoperability that is suitable to application in open environments, like the web. Protocols have been formalized in the simplest possible way (by means of FSA) to capture the essence of interoperability and to define a fine-grain conformance test.

The issue of conformance is widely studied in the literature in different research fields, like multi-agent systems (MAS) and service-oriented computing (SOA). In particular, in the area of MAS, in [7], [5] we have proposed two preliminary versions of the current proposal, the former, based on a trace semantics, consisting in an inclusione test, the latter, disregarding the case of different branching structures. The second technique was also adapted to web services [8]. Both works were limited to protocols with only two roles while, by means of the framework presented in this paper we can deal with protocols with an arbitrary finite number of roles. Inspired to this work the proposal in [1]: here an abductive framework is used to verify the conformance of services to a choreography with any number of roles. The limit of this work is that it does not consider the cases in which policies and roles have different branching structures. The first proposal of a formal notion of conformance in a declarative setting is due to Endriss *et al.* [11], the authors, however, do not prove any relation between their definitions of conformance and interoperability. Moreover, they consider protocols in which two partners strictly alternate in uttering messages.

In the SOA research field, conformance has been discussed by Foster *et al.* [12], who defined a system that translates choreographies and orchestrations in labeled transition systems so that it becomes possible to apply model checking techniques and verify properties of theirs. In particular, the system can check if a service composition complies with the rules of a choreography by equivalent interaction traces. Violations are highlighted back to the engineer. Once again, as we discussed, basing on traces can be too much restrictive. In [9], instead, "conformability bisimulation" is defined, a variant of the notion of bisimulation. This is the only work that we have found in which different branching structures are considered but, unfortunately, the test is too strong. In fact, with reference to Figure 2, it excludes the cases (b) and (c), and it also excludes cases (a) and (d) from Figure 3, which do not compromise interoperability. A recent proposal, in this same line, is [18], which suffers of the same limitations.

1) Acknowledgements.: This research has partially been funded by the European Commission and by the Swiss Federal Office for Education and Science within the 6th Framework Programme project REWERSE number 506779 (cf. http://rewerse.net), and it has also been supported by MIUR PRIN 2005 "Specification and verification of agent interaction protocols" national project.

References

[1] M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, and M. Montali, "An abductive framework for a-priori verification of web services," in *Principles and Practice of Declarative Programming, PPDP'06).* ACM Press, 2006.
[2] M. Alberti, D. Daolio, P. Torroni, M. Gavanelli, E. Lamma, and P. Mello, "Specification and verification of agent interaction protocols in a logic-based system," in *ACM SAC 2004.* ACM, 2004, pp. 72–78.
[3] G. Alonso, F. Casati, H. Kuno, and V. Machiraju, *Web Services.* Springer, 2004.
[4] A. Arnold, *Finite Transition Systems.* Pearson Education, 1994.

[5] M. Baldoni, C. Baroglio, A. Martelli, and Patti, “Verification of protocol conformance and agent interoperability,” in *Post-Proc. of CLIMA VI*, ser. LNCS State-of-the-Art Survey, vol. 3900. Springer, 2006, pp. 265–283.

[6] M. Baldoni, C. Baroglio, A. Martelli, and V. Patti, “Reasoning about interaction protocols for customizing web service selection and composition,” *J. of Logic and Alg. Progr., special issue on Web Services and Formal Methods*, 2006, to appear.

[7] M. Baldoni, C. Baroglio, A. Martelli, V. Patti, and C. Schifanella, “Verifying protocol conformance for logic-based communicating agents,” in *Proc. of CLIMA V*, ser. LNCS, no. 3487. Springer, 2005, pp. 192–212.

[8] ——, “Verifying the conformance of web services to global interaction protocols: a first step,” in *Proc. of WS-FM 2005*, ser. LNCS. Springer, September, 2005, vol. 3670, pp. 257–271.

[9] N. Busi, R. Gorrieri, C. Guidi, R. Lucchi, and G. Zavattaro, “Choreography and orchestration: a synergic approach for system design,” in *Proc. of 4th International Conference on Service Oriented Computing (ICSOC 2005)*, 2005.

[10] B. Eckel, *Thinking in Java*. Prentice Hall, 2005.

[11] U. Endriss, N. Maudet, F. Sadri, and F. Toni, “Logic-based agent communication protocols,” in *Advances in agent communication languages*, ser. LNAI, vol. 2922. Springer-Verlag, 2004, pp. 91–107, invited contribution.

[12] H. Foster, S. Uchitel, J. Magee, and J. Kramer, “Model-based analysis of obligations in web service choreography,” in *Proc. of IEEE International Conference on Internet&Web Applications and Services 2006*, 2006.

[13] R. v. Glabbeek, “Bisimulation,” Encyclopedia of Distributed Computing (J.E. Urban & P. Dasgupta, eds.), Kluwer, 2000, available at `http://Boole.stanford.edu/pub/DVI/bis.dvi.gzz`.

[14] F. Guerin and J. Pitt, “Verification and Compliance Testing,” in *Communication in Multiagent Systems*, ser. LNAI, H. Huget, Ed., vol. 2650. Springer, 2003, pp. 98–112.

[15] G. J. Holzmann, *The SPIN Model Checker : Primer and Reference Manual*. Addison-Wesley Professional, 2003.

[16] R. Milner, *Communication and Concurrency*. Prentice Hall, 1989.

[17] OMG, “Unified modeling language: Superstructure,” 2005.

[18] X. Zhao, H. Yang, and Z. Qui, “Towards the formal model and verification of web service choreography description language,” in *Proc. of WS-FM 2006*, 2006.

Importing Agent-like Interaction in Object Orientation

Matteo Baldoni, Guido Boella
Dipartimento di Informatica
Università degli Studi di Torino
C.so Svizzera, 185 — I-10149 Torino (Italy)
Email: {baldoni,guido}@di.unito.it

Leendert van der Torre
Department of Computer Science FTST
6, rue Richard Coudenhove - Kalergi
L-1359 Luxembourg
Email: leon.vandertorre@uni.lu

***Abstract*— This paper begins with the comparison of the message-sending mechanism, for communication among agents, and the method-invocation mechanism, for communication among objects. Then, we describe an extension of the method-invocation mechanism by introducing the notion of "sender" of a message, "state" of the interaction and "protocol" using the notion of "role", as it has been introduced in the** `powerJava` **extension of Java. The use of roles in communication is shown by means of an example of protocol.**

I. Introduction

The major differences of the notion of agent w.r.t. the notion of object are often considered to be "autonomy" and "proactivity" [27]. Less attention has been devoted to the peculiarities of the *communication capabilities* of agents, which exchange messages while playing roles in protocols. For example, in the contract net protocol (CNP) an agent in the role of initiator starts by *asking for* bids, while agents playing the role of participants can *propose* bids which are either *accepted* or *rejected* by the Initiator.

The main features of communication among agents which emerge from the CNP example are the following:

1) The message identifies both its *sender* and its *receiver*. E.g., in FIPA the acceptance of a proposal is:
```
(accept-proposal :sender i :receiver
j :in-reply-to
    bid089 :content X :language
FIPA-SL).
```
2) The interaction with each agent is associated to a *state* which evolves according to the messages that are exchanged. The meaning of the messages is influenced by the state. E.g., in the FIPA iterated contract net protocol, a "call for proposal" is a function of the previous calls for proposals, i.e., from the session.
3) Messages are produced according to some *protocol* (e.g., a call for proposal must be followed by a proposal or a reject).
4) The sender and the receiver play one of the *roles* specified in the protocol (e.g., initiator and participant in the contract net protocol).
5) Communication is *asynchronous*: the response to a message does not necessarily follow it immediately. E.g., in the contract net protocol, a proposal must follow a call for proposal and it must arrive, no matter when, before a given deadline.
6) The receiver autonomously decides to comply with the message (e.g., making a proposal after a call for proposal).

The message metaphor has been originally used also for describing method calls among objects, but it is not fully exploited. In particular, message-exchange in the object oriented paradigm has the following features:

1) The message is sent to the receiver without any information concerning the sender.
2) There is no state of the interaction between sender and receiver.
3) The message is independent from the previous messages sent and received.
4) The sender and the receiver do not need to play any role in the message exchange.
5) The interaction is synchronous: an object waits for the result of a method invocation.
6) The receiver always executes the method invoked if it exists.

These two scenarios are rather different but we believe that the object-oriented (OO) paradigm can learn something from the agent-oriented world. The research question of this paper is thus: is it profitable to introduce in the OO paradigm concepts taken from agent communication? how can we introduce in the OO paradigm the way agents communicate? And as subquestions: which of the above properties can be imported and which cannot? How to translate the properties which can be imported in the OO paradigm? What do we learn in the agent-oriented world from this translation?

The methodology that we use in this paper is to map the properties of agent communication to an extension of Java, `powerJava` [4], [3], [5], which adds roles to objects. Roles are used to represent the sender of a message (also known as the "player of the role"), to represent the state of the interaction via role instances, allowing the definition of protocols and asynchronous communication as well as the representation of the different relations between objects.

The choice of the Java language is due to the fact that it is one of the prototypical OO programming languages; moreover, MAS systems are often implemented in Java and some agent programming languages are extensions of Java, e.g., see the Jade framework [8] or the JACK software tool [26]. In this way we can directly use complex interaction and roles offered by our extension of Java when building MAS systems or extending agent programming languages.

Furthermore, we believe that in order to contribute to the success of the Autonomous Agents and Multiagent Systems research, the theories and concepts developed in this area should be applicable also to more traditional views. It is a challenge for the agent community to apply its concepts outside strictly agent-based applications. The OO paradigm is central in Computer Science and, as observed and suggested also by Juan and Sterling [19], before AO can be widely used in industry, its attractive theoretical properties must be first translated to simple, concrete constructs and mechanisms that are of similar granularity as objects.

The paper is organized as follows. In Section II we show which properties of agent communication can be mapped to objects. In Section III we introduce how we model interaction in `powerJava` and in Section IV we discuss how to use roles in order to model complex forms of interaction between object inspired by agent interaction, we also illustrate the contract net protocol among objects using `powerJava`. Conclusions end the paper.

II. Communication between objects

When approaching an extension of a language or of a method, the first issue that should be answered is whether that extension brings along some advantages. In our specific case, the question can be rephrased as: Is it useful for the OO paradigm to introduce a notion of communication as developed in MAS? We argue that there are several acknowledged limitations in OO method invocation which could be overcome, thus realizing what we could call a "session-aware interaction".

First of all, objects exhibit only one state in all interactions with any other object. The methods always have the same meaning, independently of the identity or type of the object from which they are called.

Second, the operational interface of Abstract Data Types induces an *asymmetrical* semantic dependency of the callers of operations on the operation provider: the caller *takes the decision* on what operation to perform and it relies on the provider to carry out the operation. Moreover, method invocation does not allow to reach a minimum level of "control from the outside" of the participating objects [2].

Third, the state of the interaction is not maintained and methods always offer the same behavior to all callers under every circumstance. This limit could be circumvented by passing the caller as a further parameter to each method and by indexing, in each method, the possible callers.

Finally, even though asynchronous method calls can be simulated by using buffers, it is still necessary to keep track of the caller explicitly.

The above problems can be solved by using the way communication is managed between agents and defining it as a primitive of the language. By adopting agent-like communication, in fact, the properties presented in Section I – with the only exception of autonomy, (6), which is a property distinguishing agents from objects – can be rewritten as in the following:

1) When methods are invoked on an object also the object invoking the method (the "sender") must be specified.
2) The state of the interaction between two objects must be maintained.
3) In presence of state information, it is possible to implement interaction protocols because methods are enabled to adapt their behavior according to the interaction that has occurred so far. So, for instance, a proposal method whose execution is not preceded by a call for proposals can detect this fact and raise an exception.
4) The object whose method is invoked and the object invoking the method play each one of the roles specified by the other, and they respect the *requirements* imposed on the roles. Intuitively, requirements are the capabilities that an object must have in order to be able to play the role.
5) The interaction can be asynchronous, thanks to the fact that the state of the interaction is maintained.

For a better intuition, let us consider as an example the case of a simple interaction schema which accounts for two objects. We expect the first object to wait for a "call for proposal" by the other object; afterwards, it will invoke the method "propose" on the caller. The idea is that the call for proposal can be performed by different callers and, depending on the caller, a different information (e.g. the information that it can understand) should be returned by the first object. More specifically, we can, then, imagine to have an object `a`, which exposes a method `cfp` and waits for other objects to invoke it. After such a call has been performed, the object `a` invokes a method `propose` on the caller. Let us suppose that two different objects, `b` and `c`, do invoke `cfp`. We desire the data returned by `a` to be different for the two callers.

Since we look at the agent paradigm the solution is to have two different interaction states, one for the interaction between `a` and `b` and one for the interaction between `a` and `c`. In our terminology, `b` and `c` interact with `a` in two distinct roles (or better, *role instances*) which have distinct states: thus it is possible to have distinct behaviors depending on the invoker. If the next move is to "accept" a proposal, then we must be able to associate the acceptance to the right proposal.

In order to implement these properties we use the notion of role introduced in the `powerJava` language in a different way with respect to how it has been designed for.

III. Modelling interaction with `powerJava`

In [24], [1], [13], [22] the concept of "role" has been proved extremely useful in programming languages for several reasons. These reasons range from dealing with the separation of concerns between the core behavior of an object and its interaction possibilities, reflecting the ontological structure of domains where roles are present, from modelling dynamic changes of behavior in a class to fostering coordination among components. In [4], [3], [5] the language `powerJava` is introduced: `powerJava` is an extension of the well-known Java language, which accounts for roles, defined within social entities like institutions, organizations, normative systems, or groups [6], [15], [28]. The name `powerJava` is due to the fact that the key feature of the proposed model is that institutions use roles to supply the *powers* for acting (*empowerment*). In particular, three are the properties that characterize roles, according to the model of normative multiagent systems [10], [11], [12]:

> **Foundation**: a (instance of) role must always be associated with an instance of the institution it belongs to (see Guarino and Welty [17]), besides being associated with an instance of its player.
> **Definitional dependence**: The definition of the role must be given inside the definition of the institution it belongs to. This is a stronger version of the definitional dependence notion proposed by Masolo *et al.* [20], where the definition of a role must include the concept of the institution.
> **Institutional empowerment**: the actions defined for the role in the definition of the institution have access to the state and actions of the institution and to the other roles' state and actions: they are powers.

Roles require to specify both *who can play the role* and *which powers are offered* by the institution in which the role is defined. The objects which can play the role might be of different classes, so that roles can be specified independently of the particular class playing the role. For example a role customer can be played both by a person and by an organization. Role specification is a sort of double-sided interface, which specifies both the methods required to a class playing the role (*requirements*, keyword "playedby") and the methods offered to objects playing the role (*powers* keyword "role"). An object, which plays a role, is empowered with new methods as specified by the interface.

To make an example, let us suppose to have a printer which supplies two different ways of accessing to it: one as a normal user, and the other as a superuser. Normal users can print their jobs and the number of printable pages is limited to a given maximum. Superusers can print any number of pages and can query for the total number of prints done so far. In order to be a user one must have an account which is printed on the pages. The role specification for the user is the following:

```
role User playedby AccountedPerson {
  int print(Job job);
  int getPrintedPages();
}

interface AccountedPerson {
  Login getLogin();
}
```

The superuser, instead:

```
role SuperUser playedby AccountedPerson {
  int print(Job job);
  int getTotalPrintedPages();
}
```

Requirements must be implemented by the objects which act as players.

```
class Person implements AccountedPerson {
  Login login;  // ...
  Login getLogin() {
    return login;
  }
}
```

Instead, powers are implemented in the class defining the institution in which the role itself is defined. To implement roles inside an institution we revise the notion of *Java inner class*, by introducing the new keyword `definerole` instead of `class` followed the name of the role definition that the class is implementing.

```
class Printer {
  final static int MAX_PAGES_PER_USER;
  private int totalPrintedPages = 0;

  private void print(Job job, Login login) {
    totalPrintedPages += job.getNumberPages();
    // performs printing
  }

  definerole User {
    int counter = 0;
    public int print(Job job) {
      if (counter > MAX_PAGES_USER)
        throws new IllegalPrintException();
      counter += job.getNumebrPages();
      Printer.this.print(job, that.getLogin());
      return counter;
    }
    public int getPrintedPages(){
      return counter;
    }
  }

  definerole SuperUser {
    public int print(Job job) {
      Printer.this.print(job, that.getLogin());
      return totalPrintedPages;
    }
    public int getTotalPrintedpages() {
      return totalPrintedPages;
    }
  }

}
```

Roles cannot be implemented in different ways in the same institution and we do not consider the possibility of extending role implementations (which is, instead, possible with inner classes), see [5] for a deeper discussion.

As a Java inner class, a role implementation has access to the private fields and methods of the outer class (in the above example the private method *print* of *Printer* used both in role *User* and in role *SuperUser*) and of the other roles defined in the outer class. This possibility does not disrupt the encapsulation principle since all roles of an institution are defined by who defines the institution itself. In other words, an object that has assumed a given role, by means of it, has access and can change the state of the corresponding institution and of the sibling roles. In this way, we realize the powers envisaged by our analysis of the notion of role.

The class implementing the role is instantiated by passing to the constructor an instance of an object satisfying the requirements. The behavior of a role instance depends on the player instance of the role, so in the method implementation the player instance can be retrieved via a new reserved keyword: `that`, which is used only in the role implementation. In the example the invocation of `that.getLogin()` as a parameter of the method `print`.

All the constructors of all roles have an implicit first parameter which must be passed as value the player of the role. The reason is that to construct a role we need both the institution the role belongs to (the object the construct `new` is invoked on) and the player of the role (the first implicit parameter). For this reason, the parameter has as its type the requirements of the role. A role instance is created by means of the construct `new` and by specifying the name of the "inner class" implementing the role which we want to instantiate. This is like it is done in Java for inner class instance creation. Differently than other objects, role instances do not exist by themselves and are always associated to their players.

Methods can be invoked from the players, given that the player is seen in its role. To do this, we introduce the new construct

receiver `<-(`*role*`)` *sender*

This operation allows the sender (player of the role) to use the powers given by "role" when it interacts with the receiver (institution) the role belongs to. It is similar to *role cast* as introduced in [3], [4], [5] but it stresses more strongly the interaction aspect of the two involved objects: the sender uses the role defined by the receiver for interacting with it. Let us see how to use this construct in our running example. The first instructions in the main create a printer object `hp8100` and two person objects, `chris` and `sergio`. `chris` is a normal user while `sergio` is a superuser. Indeed, instructions four and five define the roles of these two objects w.r.t. the created printer. The two users invoke method `print` on `hp8100`. They can do this because they have been empowered of printing by their roles. The act of printing is carried on by the private method `print`. Nevertheless, the two roles of `User` and `SuperUser` offer two different way to interact with it: `User` counts the printed pages and allows a user to print a job if the number of pages printed so far is less than a given maximum; `SuperUser` does not have such a limitation. Moreover, `SuperUser` is empowered also for viewing the total number of printed pages. Notice that the page counter is maintained in the role state and persists through different calls to methods performed by a same sender/player towards the same receiver/institution as long as it plays the role.

```
 class PrintingExample {
  public static void main(String[] args) {

    Printer hp8100 = new Printer();
    Person chris = new Person();
    Person sergio = new Person();

    hp8100.new User(chris);
    hp8100.new SuperUser(sergio);

    (hp8100 <-(User) chris).print(job1);
    (hp8100 <-(SuperUser) sergio).print(job2);
    (hp8100 <-(User) chris).print(job3);

    System.out.println("Chris has printed " +
      (hp8100 <-(User) chris).getPrintedPages()
           + " pages");
    System.out.println("The printer hp8100 has
       printed a total of " +
       (hp8100 <-(User)sergio).getTotalPrintedPages()
       + " pages");

  }
}
```

By maintaining a state, a role can be seen as realizing a *session-aware interaction*, in a way that is analogous to what done by cookies or Java sessions for JSP and Servlet. So in our example, it is possible to visualize the number of currently printed pages, as in the above example. Note that, when we talk about playing a role we always mean playing a role instance (or *qua individual* [20] or *role enacting agent* [14]) which maintains the properties of the role.

An object has different (or additional) properties when it plays a certain role, and it can perform new activities, as specified by the role definition. Moreover, a role represents a specific state which is different from the player's one, which can evolve with time by invoking methods on the roles. The relation between the object and the role must be transparent to the programmer: it is the object which has to maintain a reference to its roles. However, a role is not an independent object, it is a facet of the player.

Since an object can play multiple roles, the same method will have a different behavior, depending on the role which the object is playing when it is invoked. It is sufficient to specify which the role of a given object, we are referring to, is. In the example `chris` can become also `superuser` of `hp8100`, besides being a normal `user`

```
    hp8100.new SuperUser(chris);
    (hp8100 <-(SuperUser) chris).print(job4);
    (hp8100 <-(User) chris).print(job5);
```

Notice that in this case two different sessions will be kept: one for `chris` as normal `user` and the other for `chris` as `superuser`. Only when it prints its jobs as a normal `user` the page counter is incremented.

IV. Uses of roles in `powerJava`

In this paper we exploit the language `powerJava` in a new way which allows modelling the agent inspired vision of interaction among objects. The basic idea of `powerJava` is that objects (e.g. `hp8100`), called institutions, are composed of roles which can access the state of the institution and of other sibling roles and, thus, can coordinate with each other [4]. However, since an institution is just an object which happens to contain role implementations, nothing prevents us to consider *every object* as an institution, and to consider the roles as different ways of interacting with it. Many objects can play the same role (a printer can have many users) as well as the same object can play different roles (`chris` is both a user and a superuser). Each role instance has its own state, which represents the state of the interaction with the player of the role.

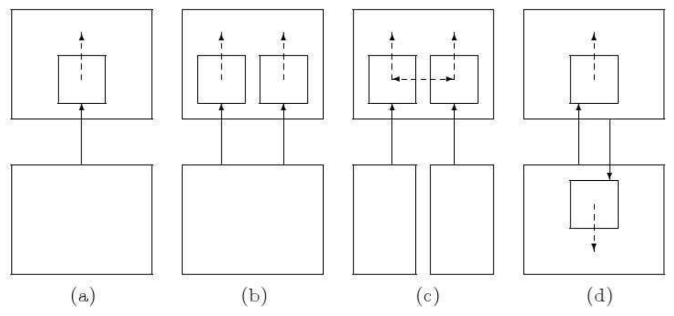


Fig. 1. The possible uses of roles.

Figure 1 illustrates the different interaction possibilities given by roles, which do not exclude the traditional direct interaction with the object when roles are not necessary. Other possibilities like sessions shared by multiple objects are not considered for space reasons.

Arrows represent the relations between players and their respective roles, *dashed arrows* represent the access relation between objects, i.e., their powers.

- Drawing (a) illustrates the situation where an object interacts with another one by means of the role offered by it. This is, for instance, the case of `sergio` being a `SuperUser` of `hp8100`.
- Drawing (b) illustrates an object (e.g., `chris`) interacting in two different roles with another one (`hp8100` in the example). This situation is used when an object implements two different interfaces for interacting with it, which have methods (like `print`) with the same signature but with different meaning. In our model the methods of the interfaces are implemented in the roles offered by the objects to interact with them. The role represent also the different sessions of the interaction with the different objects.
- Drawing (c) illustrates the case of two objects which interact by means of the roles of an institution (which can be considered as the context of execution). This is the original case, `powerJava` has been developed for [4]; in this paper, we used as a running example the well-known 5 philosophers scenario. The institution is the table, at which philosophers are sitting and coordinate to take the chopsticks and eat since they can access the state of each other. The coordinated objects are the players of the role `chopstick` and `philosopher`. The former role is played by objects which produce information, the latter by objects which consume them. None of the players contains the code necessary to coordinate with the others, which is supplied by the roles.
- In drawing (d) two objects interact with each other, each playing a role offered by the other. This is often the case of interaction protocols: e.g., an object can play the role of *initiator* in the Contract Net Protocol if and only if the other object plays the role of *participant*. Indeed, the Contract Net Protocol is reported as an example in the following section.

The four cases can be combined to represent more complex interaction schemas.

This view of roles inspires a new vision of the the OO paradigm, whose object metaphor has been accepted too acritically and it has not been subject to a deep analysis. In particular, it is a naive view of the notion of object and it does not consider the analysis of the way humans conceptualize objects performed in philosophy and above all in cognitive science [16]. In particular, cognitive science has highlighted that properties of objects are not objective properties of the world, but they depend on the properties of the agent conceptualizing the object: objects are conceptualized on the basis of what they "afford" to the actions of the entities interacting with them. Thus, different entities conceptualize the same object in different ways. We translate this intuition in the fact that an object offers different methods according to which type of object it is calling it: the methods offered (the powers of a role) depend on the requirements offered by the caller.

A. The Contract Net Protocol example

Hereafter, we report an example set in the framework of interaction protocols, describing an implementation of the well-known *contract net* protocol. The example follows the interaction schema (d), reported in the previous section, and it is substantially different than the analogous example reported in a previous paper [3]. In fact, the solution proposed here is *distributed* instead of being *centralized* (let us denote by this name a solution respecting case (c) in the previous section). The advantage of the old solution was that players did not need to know anything about the coordination mechanism. In this case, instead, each object also supplies a role for its counterpart, which describes the powers that are given to the counterpart in the interaction. For instance, the object that will play the `initiator` role will define the powers of the `participants`, and vice versa. The powers are the messages that the `initiator` will understand; this is very different than our previous proposal, where the powers only allowed to start a negotiation or to take part to a negotiation, depending on the role, and the exchanged messages were hidden inside the institution.

In this new version, roles are also used for maintaining interaction sessions. In the following example, `refuseProposal` can be executed only if `cfp` has already been executed, this can be tracked thanks to the role state and, in particular, thanks to variable `state` (set to the constant value `STATE_1` or $STATE\_2$ to check which operations of the role can be called in which state and under which condition).

Observe that when the object, offering a role, is supposed to answer something, it needs to invoke a method, which is supplied as a power of a role, which is in turn offered by the object to which it is responding. In the contract net, a possible answer to a `cfp` is the performative `propose`. In this case, see also the code reported at the end of this section, the above interaction is implemented by the instruction:

```
          (that <-(Participant)
   Peer.this).propose(getProposal(task))
```

Here, `Peer.this` refers to the object offering the role `initiator`; such an object means to play the role of `Participant` and, in particular, to invoke the power `propose` offered by this role. The role `participant` is offered by the object which is currently playing the initiator (identified in the above code line by `that`), see Fig. 2.

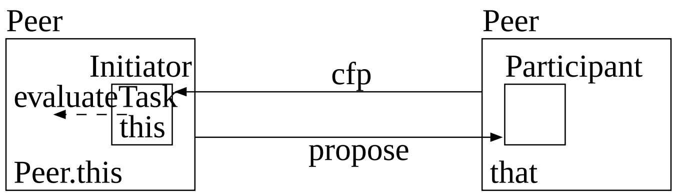


Fig. 2. Description of the interaction between an Initiator and a Participant, when, after a "cfp" performative, the answer will be a "propose" performative.

The communication is asynchronous, since the proposal is not returned by the `cfp` method.

Notice that an object which is currently playing the role of participant in a given interaction, can at the same time play the role of initiator in another interaction. See the method `evaluateTask`, in which a new interaction is started for executing a subtask by creating the two roles in the respective objects and by linking players to them:

```
role Initiator playedby InitiatorReq {
  void cfp(Task task);
  void rejectProposal(Proposal proposal);
  void acceptProposal(Proposal proposal);
}
interface InitiatorReq {
  // must implement the role specification
  // Participant
}

role Participant playedby ParticipantReq {
  void propose(Proposal proposal);
  void refuse(Task task);
  void inform(Object result);
  void failure(Object error);
}

interface ParticipantReq {
  // must implement the role specification
  // Initiator
}

class Peer implements ParticipantReq,
  InitiatorReq
{

  definerole Initiator {
    final static int STATE_1 = 1;
    final static int STATE_2 = 2;
    int state = STATE_1;

    public void cfp(Task task) {
      if (state != STATE_1)
        throws new IllegalPerfomativeException();
      state = STATE_2;
      if (evaluateTask(task))
        (that <-(Participant) Peer.this).
              propose(getProposal(task));
      else
        (that <-(Participant) Peer.this).
              refuse(task);
    }

    public void refuseProposal(Proposal proposal) {
      if (state != STATE_2)
        throws new IllegalPerformativeException();
      removeProposal(proposal);
      state = STATE_1;
    }

    public void acceptProposal(Proposal proposal) {
      if (state != STATE_2)
        throws new IllegalPerfomativeException();
      try {
        (that <-(Participant) Peer.this).
              inform(performTask(proposal, task));
      } catch(TaskExecException err) {
        (that <-(Participant) Peer.this).
              failure(err);
      }
      state = STATE_1;
    }

  }

  private boolean evaluateTask(Task task) {
    Task subTask;  // ...
    this.new Participant(peer);
    peer.new Initiator(this);
    (peer <-(Initiator) this).cfp(subTask);
       // ...
  }

  definerole Participant { ... }

}
```

V. Conclusion

In this work, we have proposed the introduction of a form of interaction between objects, in the OO paradigm, which borrows from the theory about agent communication. The main advantage is to allow session-aware interactions in which the history of the occurred method invocations can be taken into account and, thus, introducing the possibility of realizing, in a quite natural way, agent interaction protocols. The key concept which allows communication is the role played by an object in the interaction with another object. Besides proposing a model that describes this form of interaction, we have also proposed an extension of the language `powerJava` that accounts for it.

One might wonder whether the introduction of agent-like communication between objects gives us some feedback to the agent world. We believe that the following lessons can be learnt, in particular, concerning roles:

- Roles must be distinguished in role types and role instances: role instances must be related to the concept of session of an interaction.
- The notion of role is useful not only for structuring institutions and organizations but for dealing with interaction among agents.
- The notion of affordance can be used to allow agents to interacts in different ways with different kind of agents.

In this paper, we show a different way of using `powerJava` exploiting roles to model communications where: the method call specifies the caller of the object, the state of the interaction is maintained, methods can be part of protocols, objects play roles in the interaction and method calls can be asynchronous as in agent protocols.

This proposal builds upon the experience that the authors gathered on the language `powerJava` [4], [3], [5], which is implemented by means of a precompiler. Basically `powerJava` shares the idea of gathering roles inside wider entities with languages like Object Teams [18] and Ceasar [21]. These languages emerge as refinements of aspect oriented languages aiming at resolving practical limitations of other languages. In contrast, our language starts from a conceptual modelling of roles and then it implements the model as language constructs. Differently than these languages we do not model aspects. The motivation is that we want to stick as much as possible to the Java language. However, aspects can be included in our conceptual model as well, under the idea that actions of an agent playing a role "count as" actions executed by the role itself. In the same way, the execution of methods of an object can give raise by advice weaving to the execution of a method of a role. On the other hand, these languages do not provide the notion of role casting we introduce in `powerJava`. Roles as double face interfaces have some similarities with Traits [23] and Mixins [9]. However, they are distinguished because roles are used to extend instances and not classes. Finally, C# allows for multiple implementations of interfaces. None of the previous works, however, considers the fact that roles work as sessions of the interaction between objects.

Some patterns partially address the same problems of this paper. For example, the strategy design pattern allows to dynamically change the implementation of a method. However, it is complex to implement and it does not address the problem of having different methods offered to different types of callers and of maintaining the state of the interaction between caller and callee.

Baumer *et al.* [7] propose the role object pattern to solve the problem of providing context specific views of the key abstractions of a system. They argue that different context-specific views cannot be integrated in the same class, otherwise the class would have a bloated interface, and unanticipated changes would result in recompilations. Moreover, it is not possible either to consider two views on an object as an object belonging to two different classes, or else the object would not have a single identity. They propose to model context-specific views as role objects which are dynamically attached to a core object, thus forming what they call a subject. This adjunct instance should share the same interface as the core object. Our proposal is distinguished by the fact that roles are always roles of an institution. As a consequence they do not consider the additional methods of the roles as powers which are implemented using also the requirements of the role. Finally, in their model, since the role and its player share the same interface, it is not possible to express roles as partial views on the player object.

In UML 2.0 the Protocol State Machine (PSM) was introduced to specify which operations of the classifier can be called in which state and under which condition, thus specifying the allowed call sequences on the classifiers operations [25]. This diagram is particular useful for representing protocols between objects. Our proposal could help to implement a PSM because of the possibility represents directly by the notion of role all the message exchanges relevant to the protocol, increasing the readability and reusability of the protocol implementation itself.

By implementing agent like communication in an OO programming language, we gain in simplicity in the language development, importing concepts that have been developed by the agent community inside the Java language itself. This language is, undoubtedly, one of the most successful currently existing programming languages, which is also used to implement agents even though it does not supply specific features for doing it. The language extension that we propose is a step towards the overcoming of these limits.

At the same time, introducing theoretically attractive agent concepts in a widely used language can contribute to the success of the Autonomous Agents and Multiagent Systems research in other fields. Developers not interested in the complexity of agent systems can anyway benefit from the advances in this area by using simple and concrete constructs in a traditional programming language.

Future work concerns making explicit the notion of state of a protocol so to make it transparent to the programmer and allow to define the same method with different meanings in each state. Finally, the integration of centralized and decentralized approaches to coordination among roles (drawings (c) and (d) of Figure 1) must be studied.

REFERENCES

[1] A. Albano, R. Bergamini, G. Ghelli, and R. Orsini, "An object data model with roles," in *Procs. of VLDB'93*, 1993, pp. 39–51.

[2] F. Arbab, "Abstract behavior types: A foundation model for components and their composition," in *Formal Methods for Components and Objects, LNCS 2852*. Berlin: Springer Verlag, 2003, pp. 33–70.

[3] M. Baldoni, G. Boella, and L. van der Torre, "Bridging agent theory and object orientation: Importing social roles in object oriented languages," in *Procs. of PROMAS'05 workshop at AAMAS'05*, 2005.

[4] ——, "Roles as a coordination construct: Introducing powerJava," in *Procs. of MTCoord'05 workshop at COORDINATION'05*, 2005.

[5] ——, "Powerjava: ontologically founded roles in object oriented programming language," in *Procs. of OOOPS Track of SAC'06*, 2006.

[6] B. Bauer, J. Muller, and J. Odell, "Agent UML: A formalism for specifying multiagent software systems," *Int. Journal of Software Engineering and Knowledge Engineering*, vol. 11(3), pp. 207–230, 2001.

[7] D. Baumer, D. Riehle, W. Siberski, and M. Wulf, “Proc. of plop’02,” in *The role object pattern*, 2002.

[8] F. Bellifemine, A. Poggi, and G. Rimassa, “Developing multi-agent systems with a FIPA-compliant agent framework,” *Software - Practice And Experience*, vol. 31(2), pp. 103–128, 2001.

[9] L. Bettini, V. Bono, and S. Likavec, “A core calculus of mixin-based incomplete objects,” in *Procs. of FOOL Workshop*, 2004, pp. 29–41.

[10] G. Boella and L. van der Torre, “Attributing mental attitudes to roles: The agent metaphor applied to organizational design,” in *Procs. of ICEC’04*. IEEE Press, 2004.

[11] ——, “A game theoretic approach to contracts in multiagent systems,” *IEEE Transactions on Systems, Man and Cybernetics - Part C*, 2006.

[12] ——, “Security policies for sharing knowledge in virtual communities,” *IEEE Transactions on Systems, Man and Cybernetics - Part A*, 2006.

[13] M. Dahchour, A. Pirotte, and E. Zimanyi, “A generic role model for dynamic objects,” in *Procs. of CAiSE’02*, ser. LNCS, vol. 2348. Springer, 2002, pp. 643–658.

[14] M. Dastani, V. Dignum, and F. Dignum, “Role-assignment in open agent societies,” in *Procs. of AAMAS’03*. New York (NJ): ACM Press, 2003, pp. 489–496.

[15] J. Ferber, O. Gutknecht, and F. Michel, “From agents to organizations: an organizational view of multiagent systems,” in *LNCS n. 2935: Procs. of AOSE’03*. Springer Verlag, 2003, pp. 214–230.

[16] J. Gibson, *The Ecological Approach to Visual Perception*. New Jersey: Lawrence Erlabum Associates, 1979.

[17] N. Guarino and C. Welty, “Evaluating ontological decisions with onto-clean,” *Communications of ACM*, vol. 45(2), pp. 61–65, 2002.

[18] S. Herrmann, “Object teams: Improving modularity for crosscutting collaborations,” in *Procs. of Net.ObjectDays*, 2002.

[19] T. Juan and L. Sterling, “Achieving dynamic interfaces with agents concepts,” in *Procs. of AAMAS’04*, 2004.

[20] C. Masolo, L. Vieu, E. Bottazzi, C. Catenacci, R. Ferrario, A. Gangemi, and N. Guarino, “Social roles and their descriptions,” in *Procs. of KR’04*. AAAI Press, 2004, pp. 267–277.

[21] M. Mezini and K. Ostermann, “Conquering aspects with caesar,” in *Procs. of the 2nd International Conference on Aspect-Oriented Software Development (AOSD)*. ACM Press, 2004, pp. 90–100.

[22] M. Papazoglou and B. Kramer, “A database model for object dynamics,” *The VLDB Journal*, vol. 6(2), pp. 73–96, 1997.

[23] N. Scharli, S. Ducasse, O. Nierstrasz, and A. Black, “Traits: Composable units of behavior,” in *LNCS, vol. 2743: Procs. of ECOOP’03*, S. Verlag, Ed., Berlin, 2003, pp. 248–274.

[24] F. Steimann, “On the representation of roles in object-oriented and conceptual modelling,” *Data and Knowledge Engineering*, vol. 35, pp. 83–848, 2000.

[25] The Object Management Group, “Unified modeling language: Superstructure,” Available at: http://www.omg.org/.

[26] M. Winikoff, “JACK - intelligent agents: An industrial strength platform,” in *Multi-Agent Programming*, R. H. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni, Eds. Berlin: Springer Verlag, 2005, pp. 175–193.

[27] M. J. Wooldridge and N. R. Jennings, “Intelligent agents: Theory and practice,” *Knowledge Engineering Review*, vol. 10, no. 2, pp. 115–152, 1995.

[28] F. Zambonelli, N. Jennings, and M. Wooldridge, “Developing multiagent systems: The Gaia methodology,” *IEEE Transactions of Software Engineering and Methodology*, vol. 12(3), pp. 317–370, 2003.

Integrating Ontology Support within AgentService

Christian Vecchiola, Alberto Grosso, Antonio Boccalatte
DIST – Department of Communications Computer and System Sciences
University of Genova
{christian, agrosso, nino}@dist.unige.it

Abstract

This paper describes the software infrastructure introduced within the AgentService framework in order to provide support for ontology design, development, and management. Ontology enriches and normalizes the interaction among agents by establishing a domain and a set of relation among objects populating that domain. A good support for ontology definitely adds value to the design and the implementation of software agents: software engineers can take advantages of the services offered by the framework to produce new ontologies and rely on them to quickly define interaction protocols which are automatically translated into state machines used by software agents.

1. Introduction

In computer science the word ontology refers to "a data model that represents a domain and is used to reason about the objects in that domain and the relations between them" [1]. Software ontologies are used in different fields of computer science such as information architecture, semantic web, and knowledge representation. In particular, the adoption of software ontologies for knowledge representation is very attractive: ontologies contribute to provide a structure, a collection of well identified concepts along with their properties, and relations among them to a given knowledge base. For these reasons, they are very useful for software agents that base their activity mainly on the interaction with peers and on reasoning about the environment. Ontologies provide a structured and efficient way to perform these tasks.

The translation of software ontology into a collection of software artifacts representing it delivers to MAS engineers a high level of abstraction helping them in defining the interactions among agents. From a practical point of view, a given software ontology establishes the content of messages exchanged among agents and provides facilities to validate them. Moreover, ontologies are a good starting point for defining interaction protocols which are the most common way to define a structured dialogue among two entities. Hence, a good support for ontology design, development, and management, definitely gives an added value to agent programming frameworks since it simplifies and empowers the activity of MAS engineers.

This paper presents the collection of software abstractions and tools integrated into AgentService [2] which provides the framework with ontology design, development, and management (hereafter ontology service). The ontology service has been designed by following the specifications provided by FIPA [3]. In the next sections we will give a brief description of AgentService and the agent model it proposes (Section 2), then we will mostly concentrate on the entire process of defining, implementing an ontology and using it to support interaction protocol design and implementation (Section 3 and 4). Conclusions will follow.

2. AgentService

In this section we will give a brief overview of the AgentService framework by pointing out only those aspects which are relevant to understand how the ontology service is integrated into the framework. Basically, we will describe the components of the framework and we will present the agent model. For a more detailed introduction please see [2].

2.1. The Framework

AgentService is a framework to implement distributed multi-agent systems. It provides support for agent design and implementation, multi-agent system implementation management and monitoring. The components which constitute the framework are the following:

- a flexible agent model through which different agent architectures can be implemented;
- a library which defines the core of the system and the basic services of the framework;
- a software environment that hosts multi-agent systems and controls their life-cycle;

- a set of programming language extensions simplifying the implementation of software agents;
- a collection of tools supporting users in designing and implementing multi-agent systems;
- complete support for ontology definition and development;
- automatic code generation for interaction protocols with ontology integration;
- a software infrastructure allowing agents to migrate among different instances of the AgentService platform;
- a set of support programs through which users can maintain and monitor multi-agent systems.

The core of the framework relies on the Common Language Infrastructure (hereafter CLI) [4] and makes the framework portable over different implementations of this specification like Mono, Rotor, and .NET. The key features of the framework are the agent platform which is a modular hosting environment for software agents and the agent model which will be investigated in the next paragraph.

2.2. The Agent Model

The framework defines a software agent an autonomous *software entity whose activity is constituted by a set of concurrent tasks and whose state is defined by a set of shared objects*. Concurrent tasks are referred as *behaviour objects* while the term *knowledge object* is used to identify the components of the agent state.

Behaviour objects encapsulates all the computational activity of a given software agent while knowledge objects define the elements composing its knowledge base. The formers can be considered as simple little programs which have their execution stack and can communicate each other by using the shared knowledge objects. Behaviour objects can access the runtime services of the agent platform and query the FIPA management agents (AMS, MTS, DF, and Ontology Agent) in order to obtain information about the environment, the community of agents and the services they offer; for example they can query the Ontology Agent in order to know which ontologies are registered in the platform and which agents are able to understand messages belonging to a given ontology.

Knowledge objects are data structures containing items which are exposed as properties. They resemble a C struct or a Pascal record, but are designed with a built support for persistence and concurrent multiple accesses. Knowledge objects define the knowledge base of a given software agent and the collection of their properties along with the execution state of each agent define the state of an agent instance.

Agents can interpret roles into a given communication protocol and they can publish this service through the DF which makes this information available to the entire community of agents. Interpreting a role makes the agent able to participate into the communication protocol which defines that role. This feature is implemented by providing the agent with a behaviour object which automatically executes the state machine defining the role. This issue will be further detailed in the next section.

3. Ontology Support in AgentService

AgentService provides a complete support to ontology design, implementation, and management. These three functionalities are collectively referred as the *ontology service*. The ontology service is based on the following framework components:

- a set of classes representing the object model defining all the elements required to represent an ontology (classes, concepts, instances, attributes, constraints, validation, etc);
- a set of tools that can be used to automatically generate the specific classes for a given ontology by starting from its visual or textual representation;
- an Ontology Agent (OA) which maintains the knowledge about all the ontologies registered in the hosting agent platform and about the agent which are able to communicate by using the concepts defined into a given ontology;
- FIPA SL0 [5] ACL message support.

As we can notice, from the previous list the framework does not directly provide any facility to visually design software ontologies. We decided to rely on a very well know and established tool that is Protégé [6] a software projects maintained by the KSI lab the Stanford University. Protégé, when equipped with Jambalaya [7], provides all the required features to quickly design a given ontology. In the following we will briefly illustrate the object model designed to support ontology definitions in AgentService, the role of the Ontology Agent, and the ontology development process.

3.1. Ontology Object Model

The design and the implementation of the object model defining the ontology reflects the specifications outlined in the corresponding FIPA standards [3] and has been inspired by the type system designed in JADE [8] to support ontologies. The object model defined within AgentService defines a meta-ontology which contains all the concepts and the elements which are required to compose user defined ontologies. The meta-ontology defines the following entities: *predicate*, *term*, *concept*, *query*, *action*, *variable*, *primitive*, and *aggregate*. Figure 1 describes how these elements are connected each other.

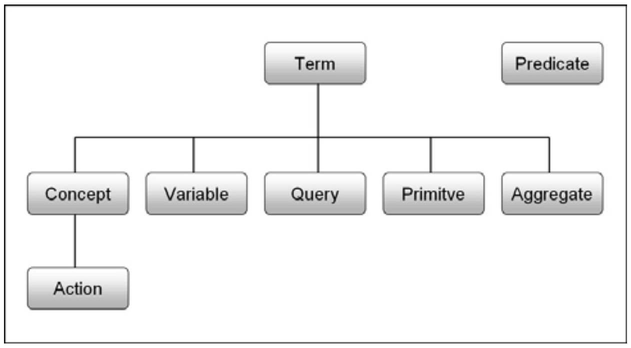


Figure 1. Ontology elements hierarchy

The elements depicted in figure 1 define the domain in which every communication based on a given ontology takes place. User defined ontologies will provide specific instances of these elements and MAS engineers will have to specialize the abstract classes representing the entities defined by the meta-ontology: the new classes will represents the concepts, the queries, the actions, the terms, etc. which are pertaining to the specific problem domain. Ontologies are also described using *schemas* which are generic objects used to describe any ontology element and provides all the information to represent a specific ontology without using ontologies specific classes. Ontology schemas are used internally by the framework in order to maintain a catalog of all the ontologies registered, while ontologies specific classes constitute the API used by software agents to hard-code the communication based on a specific ontology. Agents which can dynamically learn to use a given ontology by exploiting the services of the *OntologyDescriptor* class which automatically extracts all the useful information about a given ontology. It retrieves all the schemas defining the specific entities of the ontology and provides also other useful information such as assembly location, file versioning and type names. The implemented solution is very flexible since it provides an efficient way of using ontologies when they are statically identified, while, at the same time, provides facilities to dynamically reflect[1] ontologies.

AgentService also provides support for SL0 which is a minimal subset of the FIPA ACL message specification. The framework defines a set of classes able to represent all the elements of SL0 plus the = (equals) predicate. In order to communicate with a given ontology agents exchange messages which contains SL0 objects and a specific message content has been designed (the *ACLMessageBody* class) in order to transport SL0 statements.

The joint use of the ontology API and of SL0 as a vehicle allows agents to easily communicate each other.

3.2. The Ontology Agent

In order to be compliant with the specification provided by FIPA we have introduced the Ontology Agent which is responsible of maintaining the catalog of all the ontologies registered with the system and of providing useful information to software agents. The entire list of task that should be performed by the ontology agent is the following:

- ontology discovery and publishing;
- ontology maintenance;
- ontology mapping and translation;
- shared ontology discovery.

The ontology agent provided with the framework implements only the two features of the previous list which are also the most important. We think that the ontology mapping service is a very difficult task to implement and requires some sort of inductive knowledge in order to detect similarities among different knowledge representations.

In order to be available to the community of agents the Ontology Agent registers its service to the DF. Since we have implemented a reduced set of task of the ontology agent we decided to embed these functionalities directly into the Directory Facilitator, by adding a specific behaviour which performs these tasks. The community of agents asks to the DF which agent provides the ontology service and the directory facilitator returns its own agent identifier. Hence, the implementation of this feature is completely transparent to the community of agents which only expect to obtain the address of the Ontology Agent in order to query it.

3.3. Ontology Development Process

In order to make users feel at ease and increase their productivity we adopt, as written above, Protégé in conjunction with Jambalaya for defining AgentService ontologies. Figure 2 describes the entire process that by starting from the Protégé editor generates the assembly containing the type definitions for the ontology which are required by the AgentService framework.

Projects designed in Protégé can be exported into the XML format and a tool provided with AgentService automatically generates the corresponding object model, writes the source code and compiles it into an assembly

[1] The verb reflect is used in the sense of *type reflection* which identifies a well know set of operation aimed to extract information about the class type of a given object.

which can be easily deployed into the agent platform or used by the protocol designer. After the compilation process takes place MAS engineers are provided with the entire object model describing the ontology they designed with Protégé. The assembly can be included into a generic software project and used as a library, directly deployed into the agent platform, or, as showed in figure 2, used within the protocol designer.

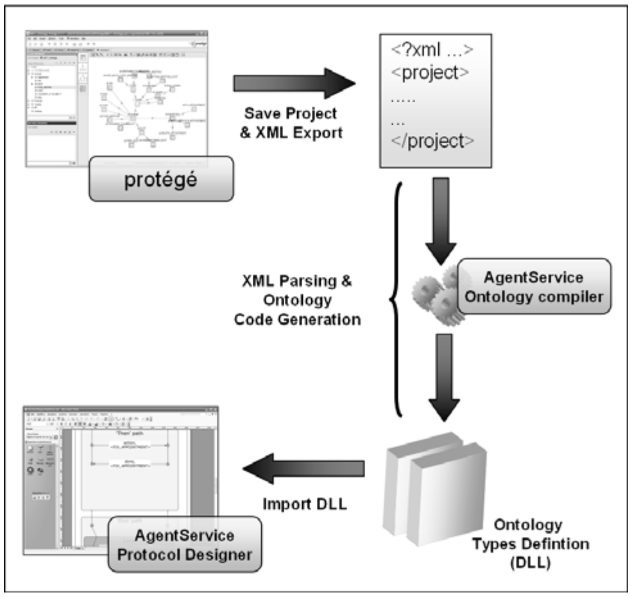

protégé
Save Project & XML Export
<?xml ...>
<project>
.....
...
</project>
XML Parsing & Ontology Code Generation
AgentService Ontology compiler
Import DLL
AgentService Protocol Designer
Ontology Types Defintion (DLL)

Figure 2. Ontology code generation

4. Protocol Design and Implementation

AgentService provide facilities to design, to implement, and to integrate interaction protocols into multi-agent systems. Interaction protocols rely on the ability of agents of communicate each other by using the services offered by the MTS and the messaging subsystem. The development of a protocol for AgentService can be enhanced by using ontologies which give a sound meaning and a well defined structure to the messages exchanged during the interaction. AgentService relies on the Microsoft Visio visual modeling environment and provides a plug-in which allows users to define interaction protocols by following the AUML standard [9].

In particular by using protocol designer developers can:

- design protocol steps for each agent role involved;
- adopt previously defined ontology for message content;
- export the protocol in a format usable by AgentService;
- design agents interpreting the roles through dedicated behaviours.

The plug-in introduces a new stencil in the Visio environment which contains all the elements to visually compose the protocol, and a toolbar which allows to automatically generate the code implementing the state machines for the protocols. Such state machines can be embedded into specific behaviour objects which are then able to participate in the interaction protocol.

The next paragraphs analyze more in details how the tool works, in particular some extensions of the AUML basic elements are introduced and the code generation process is explained.

4.1 Designing AUML elements

Agent UML proposes extensions to UML and idioms within UML in particular for sequence e collaboration diagrams. Sequence diagrams seem to be the best way to design and represent agent in order to capture inter-agent dynamics [10]. Two fundamental parts constitute the diagram model a frame, which delimits the sequence diagram, and the message flow between roles through a set of lifelines and messages [11]. Hence the frame element contains lifelines, sets of messages, and AUML operators commonly called *combined fragments*. In addition to the basic elements proposed by AUML specification (lifeline, message, alternative, optional, break, loop), we introduce some new features in order to make the model effective from the AgentService point of view.

Since the definition of the protocol is designed in order to generate running code for AgentService platform, the tool provides features for inserting lines of .NET code for each role. In particular designers can write code in order to manage exceptions during protocol execution or reply to an error message received from a peer.

Messages are obviously the core of protocol interactions; the tool provides two kinds of messages: ontological messages and native messages. The designer is integrated with the AgentService ontology system; in order to adopt an ontology it is only necessary to indicate the assembly containing it. Hence if a user wants to use an ontology within a protocol, he has to select an SL0 operator and then choose the ontological elements contained within the previously loaded ontology. In the case the user does not adopt an ontology, he has to define the fields composing the body of the message, specifying the name and the type of the message element (the message content of AgentService is strongly typed).

The tool allows developer to design *peer-to-peer* or *client-server* communications defining the cardinality of the agent roles involved in a protocol. AgentService messages are asynchronous; in client-server protocol the reception by the server is time-outed and the server manages client interaction by adopting a round-robin approach.

The AUML tool for AgentService allows users to define guard conditions (for alternative and optional statements) as Boolean expressions which very often involve elements contained within messages previously received. Hence the guard condition is not a simple label but it is a fundamental element involved in the generation of the code modeling the protocol.

Finally the AUML Loop operator is extended in order to be able to clearly indicate the agent role that manages the loop.

4.2 Code generation and execution of interaction protocols

Starting from the model representing the interaction protocol it is import to be able to generate agents playing roles within the protocol and then execute them. The designer only defines the structure of the protocol and creates classes targeting the AgentService object model which represent the state machine executing the interaction protocol. It is up to the agent playing a specific role to complete the state machine generated by the tool with all the information it needs.

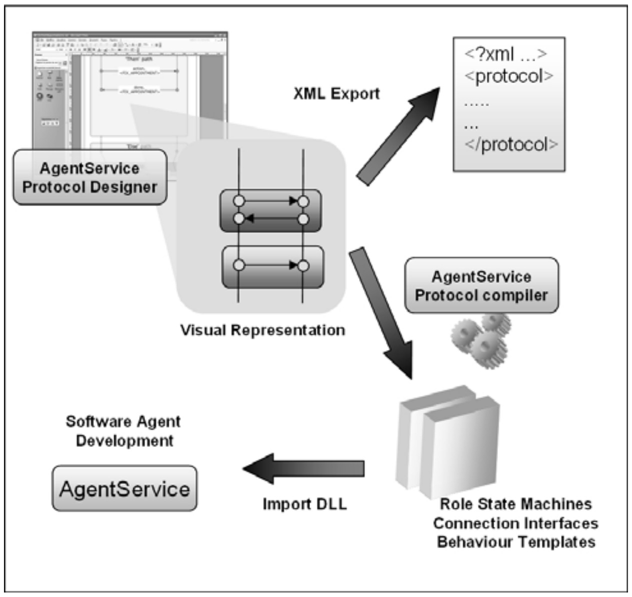


Figure 3. Protocol development process

Figure 3 describes the protocol development process: interaction protocols can either be exported into XML format or into assemblies which can be directly used to program software agents. Both of the two representations contain the same information: the first is mainly used to maintain a textual representation of the protocol and to export it to third party applications, while the second is the most important since it actually allows the use of protocols inside the framework. Given an XML or a visual representation of the interaction protocol, the protocol compiler generates an assembly containing the following entities:

- the definition of the protocol object model;
- the definition of state machines implementing the roles defined in the interaction protocol;
- the definition of the interfaces types used by the state machines to customize the execution flow of the protocol.

Software agents, in order to participate into a protocol, have to interpret one of the roles defined in the protocol. This role is interpreted by adding a behaviour object which executes the state machine defining the role; such behaviour object has to implement the interface required by the state machine and through the methods exposed by this interface interacts and controls, when possible, the execution of the state machine. The customization level provided by interface is required, for example, in order to let the agent choose among different alternatives and then drive the execution of the protocol.

The state machines automatically handle all the exceptions that can occur while executing the protocol (i.e. wrong or malformed messages, etc.) and the behaviour objects running the state machine can be informed about these exceptions through specific events.

4.2 Related works

There are some interesting works based on Agent UML involving the definition of design tools. Ehrler and Cranefield propose a Plug-in for Agent UML Linking (PAUL) [12] based on the FIPA-compliant agent platform Opal [13] and the Eclipse Modeling Framework. In particular this tool uses the UML Object Constraint Language (OCL) [14] in order to define the input and output of the operations an agent can perform within a protocol. Our design tool does not require the definition of I/O data for protocol operations; agents playing a protocol role have access to all the content of the messages received during the interaction and of course can consult their knowledge base.

Winikoff [15] precisely defines the syntax of a subset of AUML by using a textual notation; he provides a tool for designing diagrams in order to support the notation. This tool cannot generate code for the models that have to be implemented manually.

Viper [16] is a graphical editor based on the earlier version of AUML that can generate code for AgentFactory [17]. The tool, in the implementation phase, involves users for populating the protocol with customized agent code, which together with code automatically generated to reflect the protocol semantics is compiled into useable agent designs. Hence the tool generates a skeleton of code implementing the protocol and users have to complete and compile it. Instead the AgentService protocol designer generates an assembly containing the classes representing the state machine of the protocol and a programming interface that users can implement in order to customize agent operations. Viper is not provided with an ontology system.

5. Conclusions

In this paper we presented the software infrastructure introduced into the AgentService framework in order to support ontology design, implementation, and management. Software ontologies are a high level abstraction which is very useful for identifying the concepts of a problem domain, to define their relation, and to reason about them. Ontologies give an added value to the interaction among software agents since they provide facilities to define a communication and to validate messages.

The support provided by AgentService covers, either directly or not, all the activities previously cited: AgentService relies on Protégé in order to define a new ontology and translates the representation provided by Protégé into a collection of classes fitting the object model defined into the framework. These classes can be easily used to define a conversation among software agents: messages can be verified against a specific ontology and eventually discarded if spurious. By querying the ontology agent we can dynamically inspect the ontology catalog maintained in every multi-agent system and extract useful information about their structure, such information can be easily used in order to start a communication with agents which know that ontology.

Software ontologies can also be useful to define structured conversations among agents since they completely define the content of the exchanged messages. AgentService provides a useful tool to visually define interaction protocol integrated with the ontology service: it is then possible to start from the definition of the problem domain with Protégé, translate that representation into a software ontology, build an interaction protocol based on those concepts, and produce a state machine which can be directly used by software agents to interpret roles. All this process can be performed without writing a line of code thanks to the supports provide by the framework.

Nonetheless, the ontology service can be improved: now AgentService provides support only for the SL0 subset of the FIPA ACL specification while a complete implementation of the SL2 specification is still to come. Moreover, a more complete service provided by the ontology agent has to be implemented in order to be completely compliant with the FIPA specifications.

6. References

[1] Wikipedia, definition of software ontology, available at http://en.wikipedia.org/wiki/Ontology (computer sci-ence).

[2] C. Vecchiola, A. Grosso, A.Gozzi, and A. Boccalatte, “AgentService”, *Proceedings of the 16th International Conference on Software Enginnering and Knowledge Engineering (SEKE04)*, Banff, Alberta Canada, KSI Publisher, 2004.

[3] FIPA, “FIPA Ontology Service Specification”, 2001, [Online document], Availabe at HTTP:http://www.fipa.org/specs/fipa00086/XC00086D.pdf.

[4] Standard ISO/IEC 23271:2003: Common Language Infrastructure, March 28, 2003, ISO.

[5] FIPA, “FIPA SL Content Language Specification”, [Online document] available at HTTP: http://www.fipa.org/specs/fipa00008/SC00008I.html.

[6] J. Gennari, M. A. Musen, R. W. Fergerson, W. E. Grosso, M. Crubezy, H. Eriksson, N. F. Noy, and S. W. Tu, “The Evolution of Protégé: An Environment for Knowledge-Based Systems Development”, Stanford Knowledge System Lab, 2002.

[7] M. Storey, M. Musen, J. Silva, C. Best, N. Ernst, R. Fergerson, and N. Noy, “Jambalaya: Interactive visualization to enhance ontology authoring and knowledge acquisition in Protégé”, *Workshop on Interactive Tools for Knowledge Capture*, Victoria, BC, Canada, 2001.

[8] G. Caire, “Jade Tutorial – Application-defined Content Languages and Ontologies”, 2002.

[9] J. Odell, H. V.D. Parunak, and B. Bauer, “Extending UML for agents”, *Proceedings of Agent-Oriented Information Systems Workshop (AOIS-00)*, 2000.

[10] J. Odell, H. V.D. Parunak, and B. Bauer, “Representing agent interaction protocols in uml”, *Paolo Ciancarini and Michael J. Wooldridge Editors, Agent-Oriented Software Engineering*, No. 1957 in LNCS, pp. 121–140, Springer Verlag, 2000.

[11] FIPA. “FIPA Interaction Protocol Library Specification”, 2001. [Online document], Availabe at HTTP: http://www.fipa.org/specs/fipa00025/XC00025E.pdf.

[12] L. Ehrler and S. Cranefield, “Executing agent UML diagrams”, *Proceedings of the third international conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pp. 906–913, 2004.

[13] M. Purvis, S. Cranefield, M. Nowostawski, and D. Carter, "Opal: A Multi-Level Infrastructure for Agent-Oriented Software Development", *Information Science Discussion Paper Series*, No. 2002.

[14] J. B. Warmer and A. G. Kleppe, "The Object Constraint Language: Getting your models ready for MDA", *Addison-Wesley*, 2nd edition, 2003.

[15] M. Winikoff, "Towards Making Agent UML Practical: A Textual Notation and a Tool", *First international workshop on Integration of Software Engineering and Agent Technology (ISEAT 2005)*, Melbourne, Australia, 2005.

[16] C. Rooney, R. Collier, and G. M. P. O'Hare, "VIPER: Visual Protocol EditoR", *6th International Conference on Coordination Languages and Models (COORDINATION 2004)*, 2004.

[17] R. Collier, "Agent Factory: A Framework for the Engineering of Agent-Oriented Applications", *Ph.D. Thesis*, Department of Computer Science, University College Dublin, Ireland, 2001.

Collective Sorting Tuple Spaces

Matteo Casadei, Luca Gardelli, Mirko Viroli
DEIS, Cesena
Alma Mater Studiorum – Università di Bologna
via Venezia 52, 47023 Cesena (FC), Italy
{m.casadei,luca.gardelli,mirko.viroli}@unibo.it

***Abstract*— Coordination of multiagent systems is recently moving towards the application of techniques coming from the research context of complex systems: adaptivity and self-organisation are exploited in order to tackle openness, dynamism and unpredictability of typical multiagent systems applications. In this paper we focus on a coordination problem called *collective sorting*, where autonomous agents are assigned the task of moving tuples across different tuple spaces according to local criteria, resulting in the emergence of the complete clustering property. Using a library we developed for the MAUDE term rewriting system, we simulate the behaviour of this system and evaluate some solutions to this problem.**

I. INTRODUCTION

Systems that should self-organise to unpredictable changes in their environment very often need to feature adaptivity as an emergent property. As this observation was first made in the context of natural systems, it was shortly recognised as an inspiring metaphor for artificial systems as well [1]. However, a main problem with emergent properties is that, by their very definition, they cannot be achieved through a systematic design: their dynamics and outcomes cannot be fully predicted. Nonetheless, providing some design support in this context is still possible. The whole system of interest, that is the application to design and the environment it is immersed in, can be modelled as a stochastic system, namely, a system whose dynamics and duration aspects are probabilistic. In this scenario, simulations can be run and used as a fruitful tool to predict certain aspects of the system behaviour, and to support a correct design before actually implementing the application at hand [2].

This scenario is particularly interesting for agent coordination. Some works like the TOTA middleware [3], SwarmLinda [4], and stochastic KLAIM [5], though starting from different perspectives, all develop on the idea of extending standard coordination models with features related to adaptivity and self-organization. They share the idea that tuples in a tuple space eventually spread to other tuple spaces in a non-deterministic way, depending on certain timing and probability issues. Accordingly, in this paper we start analysing the potential role that simulation tools can have in this context, towards the identification of some methodological approach to system design.

As a reference example, we consider an application to a tuple space scenario of the so-called *collective sorting* problem for swarm intelligence [1]. This application features autonomous agents managing a set of distributed tuple spaces, with the goal of moving tuples from one space to the other until completely "sorting" them, that is, tuples of different types reside in different tuple spaces. We show a solution to this problem based on a fully-distributed algorithm, where each agent moves tuples according to fully-local criteria, and where complete sorting appear to emerge from initial chaotic tuple configurations. To provide evidence of correctness and appropriateness we rely on simulations.

Many simulation tools can be exploited to this end, though they all necessarily force the designer to exploit a given specification language, and therefore better apply to certain scenarios and not to others—examples are SPIM [6], SWARM [7] and REPAST [8]. Instead of relying on one of them, in this paper we seek for a general-purpose approach. We evaluate the applicability of the MAUDE specification tool as a general-purpose engine for running simulations [9]. It is very well known that MAUDE allows for modelling syntactic and dynamic aspects of a system in a quite flexible way, supporting e.g. process algebraic, automata, and net-like specifications—all of which can be seen as instantiations of MAUDE's term rewriting framework. We developed a library for allowing a system designer to specify in a custom way a system model in terms of a stochastic transition system—a labelled transition system where actions are associated with a *rate* (of occurrence) [10]. One such specification is then exploited by the tool to perform simulations of the system behaviour, thus making it possible to observe the emergence of certain (possibly unexpected) properties.

The remainder of this paper is as follows: Section 2 provides some background on coordination techniques featuring adaptivity, Section 3 describes the collective sorting problem, while Section 4 presents the MAUDE model of the Collective Sorting and its simulation results, and finally Section 5 concludes providing perspectives on future works.

II. BACKGROUND

In the effort to improve the design process of software systems—i.e. to bridge the gap between the design and the actual implementation—it has become very common practice to take into account not only functional and architectural requirements, but also quantitative aspects like temporal and probabilistic ones. When dealing with complex systems, it is often the case that aleatory in system dynamics may cause the emergence of interesting properties, that cannot therefore be abstracted away when designing the system. Coordination models and technologies for multiagent systems are witnessing the development of a number of works moving to this direction, most of which are inspired by natural phenomena.

A first example is the TOTA (Tuples On The Air) middleware [3] for pervasive computing applications, inspired by the concept of field in physics—like e.g. the gravitational or magnetic fields. This middleware supports the concept of "spatially distributed tuple": that is, a tuple can be cloned and spread to the tuple spaces in the neighborhood, creating a sort of computational field, which grows when initially pumped and then eventually fades. To this end, when injected in a tuple space, each tuple can be equipped by some application-dependent rules, defining how a tuple should spread across the network, how the content of the tuple should be accordingly affected, and so on. TOTA is mainly targeted to support multiagent systems whose environment is open, dynamic and unpredictable, like e.g. to let mobile agents meet each other in a dynamic network.

Another example is the SwarmLinda coordination model [4], which though similar to TOTA is more inspired by swarm intelligence and stigmergy [1], [11], [12]. In SwarmLinda tuples are moved from one tuple space to the other, and ant-like algorithms are used to retrieve them. The use of self-techniques in SwarmLinda derives from necessity of dealing with openness and with the unpredictability of a tuple space's users, against the need of achieving adaptivity.

Finally, the "swarm robotics" field applies strategies inspired by social insects in order to coordinate the activities of a multiplicity of robots systems. Typically, these systems are built on top of ad-hoc software middlewares [1], and solve problems with distributed-algorithms where, though each robot brings about very simple goals, the whole system can be used to solve quite complex problems—see e.g. the collective sorting problem in Section III-A.

These are all examples witnessing the fact that coordination in open, dynamic, and unpredictable systems have quantitative aspects playing a very important role. This calls for analysis and design tools that can support system development at various levels, from formal specification up to simulations.

III. Collective Sorting

A. General Scenario

We consider a case of Swarm-like intelligence known as *collective sorting* [1]. It features a multiagent system where the environment is structured and populated with items of different kinds: the goal of agents is to collect and move items across the environment so as to order them according to an arbitrary shared criterion. This problem basically amounts to clustering: homogeneous items should be grouped together and should be separated from others. Moving to a typical context of coordination models and languages, we consider the case of a fixed number of tuple spaces hosting tuples of a known set of tuple types. The goal of agents is to move tuples from one tuple space to the other until the tuples are clustered in different tuple spaces according to their tuple type.

In several scenarios, sorting tuples may increase the overall system efficiency. For instance, it can make it easier for an agent to find an information of interest based on its previous experience: the probability of finding an information where a previous and related one was found is high. Moreover, when tuple spaces contain tuples of one kind only, it is possible to apply aggregation techniques to improve their performance, and it is generally easier to manage and achieve load-balancing.

Increasing system order however comes at a computational price. Achieving ordering is a task that should be generally performed online and in background, i.e. while the system is running and without adding a significant overhead to the main system functionalities. Indeed, it might be interesting to look for suboptimum algorithms, which are able to guarantee a certain degree of ordering in time.

Nature is a rich source of simple but robust strategies: the behaviour we are looking for has already been explored in the domain of social insects. Ants perform similar tasks when organizing broods and larvae: this class of coordination strategies are generally referred to as *collective sorting* or *collective clustering* [1]. Although the actual behaviour of ants is still not fully understood, there are several models that are able to mimic the dynamics of the system. Ants wander randomly and their behaviour is modelled by two probabilities, respectively, the probability to pick up P_p and drop P_d an item

$$P_p = \left(\frac{k_1}{k_1 + f}\right)^2, \quad P_d = \left(\frac{f}{k_2 + f}\right)^2, \tag{1}$$

where k_1 and k_2 are constant parameters and f is the number of items perceived by an ant in its neighborhood: f may be evaluated with respect to the recently encountered items. To evaluate the system dynamics, apart from visualising it, it can be useful to provide a measure of the system order. Such an estimation can be obtained by measuring the spatial entropy, as done e.g. in [11]. Basically, the environment is subdivided into nodes and P_i is the fraction of items within a node, hence the local entropy is $H_i = -P_i \log P_i$. The sum of H_i having $P_i > 0$ gives an estimation of the order of the entire system, which is supposed to decrease in time, hopefully reaching zero (complete clustering).

B. An Architecture for Implementing Collective Sorting

We conceive a multiagent system as a collection of agents interacting with/via tuple spaces: agents are allowed to read, insert and remove tuples in the tuple spaces. Additionally, and transparently to the agents, an infrastructure provides a sorting service in order to maintain a certain degree of order of tuples in tuple spaces. This service is realised by a class of agents that will be responsible for the sorting task. Hence, each tuple space is associated with a pool of agents, as shown in Figure 1, whose task is to compare the content of the local tuple space against the content of another tuple space in the environment, and possibly move some tuple. Since we want to perform this task online and in background, and with a fully-distributed, swarm-like algorithm, we cannot compute the probabilities in

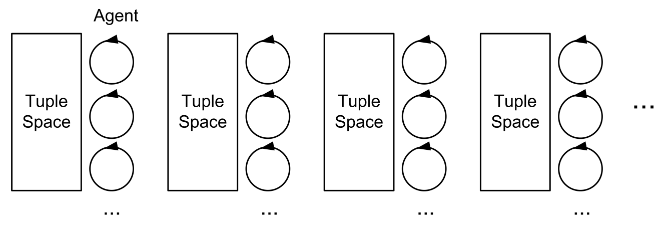


Fig. 1. The basic architecture consists in a set of sorter agents dedicated to a single tuple space.

Equation 1 to decide whether to move or not a tuple: the approach would not be scalable since it requires to count all the tuples for each tuple space, which might not be practical.

Hence, we devise a strategy based on tuple sampling, and suppose that tuple spaces provide for a reading primitive we call `urd`, *uniform read*. This is a variant of the standard `rd` primitive that takes a tuple template and yields any tuple matching the template: primitive `urd` instead chooses the tuple in a probabilistic way among all the tuples that could be returned. For instance, if a tuple space has 10 copies of tuple $t(1)$ and 20 copies of tuple $t(2)$ then the probability that operation $urd(t(X))$ returns $t(2)$ is twice as much as $t(1)$'s. As standard Linda-like tuple spaces typically do not implement this variant, it can e.g. be supported by some more expressive model like ReSpecT tuple centres [13]. When deciding to move a tuple, an agent working on the tuple space TS_S follows this agenda:

1) it draws a destination tuple space TS_D different from the source one TS_S;
2) it draws a kind k of tuple;
3) it (uniformly) reads a tuple T_1 from TS_S;
4) it (uniformly) reads a tuple T_2 from TS_D;
5) if the kind of T_2 is k and it differs from the kind of T_1, then it moves a tuple of the kind k from TS_S to TS_D.

The point of last task is that if those conditions hold, then the number of tuples k in TS_D is more likely higher than in TS_S, therefore a tuple could/should be moved. It is important that all choices are performed according to a uniform probability distribution: while in the steps 1 and 2 it guarantees fairness, in steps 3 and 4 it guarantees that the obtained ordering is appropriate.

It is worth noting that the success of this distributed algorithm is an emergent property, affected by both probability and timing aspects. Will complete ordering be reached starting from a completely chaotic situation? Will complete ordering be reached starting from the case where all tuples occur in just one tuple space? And if ordering is reached, how many moving attempts are globally necessary? These are the sort of questions that could be addressed at the early stages of design, thanks to a simulation tool.

IV. THE COLLECTIVE SORTING IN MAUDE

In this section we briefly describe a MAUDE specification of our solution to the collective sorting problem, and show simulation results. Our model sticks to the case where 4 tuple spaces exist (labelled with identifiers 0, 1, 2 and 3), and four tuple kinds are subject to ordering (`'a`, `'b`, `'c`, and `'d`).

A. A MAUDE library for simulation

MAUDE is a high-performance reflective language supporting both equational and rewriting logic specifications, for specifying a wide range of applications [9]. The basic brick of a MAUDE program is the *module*, which is essentially a set of definitions determining an algebra: the modules can be either of the *functional* or *system* kind. Functional modules contain both (syntax-customed) type and operation declarations, along with *equations* which are actually *equational rewriting* rules defining abstract data types—this is hence useful to declare algorithmic aspects of computing systems. System modules can instead have *rewriting laws* as well—i.e. transition rules—that are typically used to implement a concurrent *rewriting semantics*, and are then able to deal with aspects related to interaction and system evolution. In the course of finding a general simulation tool for stochastic systems, we find MAUDE as a particularly appealing framework, for it allows to directly model a system in terms of transition rules, or to prototype a new domain-dependent language to have more expressiveness and compact specifications.

Using MAUDE, we realized a general simulation framework for stochastic systems: the idea of this tool is to model a stochastic system by a labelled transition system where transitions are of the kind $S \xrightarrow{r:a} S'$, meaning that the system in state S can move to state S' by action a, where r is the *(global) rate* of action a in state S. The rate of an action in a given state can be understood as the number of times action a could occur in a time-unit (if the system would rest in state S), namely, its occurrence frequency. This idea is inspired by the activity mechanism of stochastic π-Calculus [10], where each channel is given a fixed local rate, and the global rate of an interaction is computed as the channel rate multiplied by the number of processes willing to send a message and the number of processes willing to receive a message. Our model is hence a generalisation of this approach, for the way the global rate is computed is custom, and ultimately depends on the application at hand—e.g. the global rate can be fixed, or can depend on the number of system sub-processes willing to execute an action. Given a transition system of this kind and an initial state, a simulation is simply executed by: *(i)* checking each time the available actions and their rate; *(ii)* picking one of them probabilistically (the higher the rate, the more likely the action should occur); *(iii)* accordingly changing the system state; and finally *(iv)* advancing the time counter according to an exponential distribution, so that the average frequency is the sum of the action rates. This technique is again a generalisation of the one adopted in the SPIM simulation engine for stochastic π-Calculus [6]. For a detailed description of the simulation framework, refer to [14].

B. The Collective Sorting model

The MAUDE specification of the Collective Sorting system is divided in three modules, respectively defining

```
mod CS is
 pr CS .  pr STANDARD-CARRIER .

 op source : Nat -> Action .              *** SYNTAX OF ACTIONS AND STATES
 op chooseTarget : -> Action .
 op chooseTupleType : -> Action .
 op readSource : -> Action .
 op readTarget : -> Action .
 op move : -> Action .

 subsort DataSpace < State .
                                          ***  A REFERNCE INITIAL STATE
 op SS : -> State .
 eq SS = ( init | < 0 @ ('a[100])|('b[100])|('c[10])|('d[10]) > |
                  < 1 @ ('a[  0])|('b[100])|('c[10])|('d[10]) > |
                  < 2 @ ('a[ 10])|('b[ 50])|('c[50])|('d[10]) > |
                  < 3 @ ('a[ 50])|('b[ 10])|('c[10])|('d[50]) > |
                  ('a , 'b , 'c , 'd ) ) .

 *** IDENTIFYING SOURCE                   ***  TRANSITION SYSTEM SEMANTICS
 eq  (init | DS)==>  =
   ( source(0) # 0.25 -> [ [0] | DS ] );
   ( source(1) # 0.25 -> [ [1] | DS ] );
   ( source(2) # 0.25 -> [ [2] | DS ] );
   ( source(3) # 0.25 -> [ [3] | DS ] ) .

 *** CHOOSING TARGET
 eq ([Ns]      | DS)   ==>  = (chooseTarget # now -> [ [Ns];[range(3)]| DS ]) .
 eq ([Ns];[Ns] | DS)   ==>  = (chooseTarget # now -> [ [Ns];[3]       | DS ]) .

 *** CHOOSING TUPLE TYPE QQ
 ceq ([Ns];[Nt]      | < Ns @ MT > | DS ) ==> = ( chooseTupleType # now -> [
     ([Ns];[Nt];[QQ] | < Ns @ MT > | DS ) ] )
        if QQ := choose(occurringTuples(MT)) .

 *** READING FROM SOURCE
 ceq ([Ns];[Nt];[Q]      | < Ns @ MT > | QL | DS ) ==> = ( readSource # now -> [
     ([Ns];[Nt];[Q];[QQ] | < Ns @ MT > | QL | DS ) ] )
        if QQ := get( QL , sample(quantities(QL, MT))) .

 *** READING FROM TARGET
 ceq ([Ns];[Nt];[Q];[Q1]      | < Nt @ MT > | QL | DS ) ==> = ( readTarget # now -> [
     ([Ns];[Nt];[Q];[Q1];[QQ] | < Nt @ MT > | QL | DS ) ] )
        if QQ := get( QL , sample (quantities(QL, MT))) .

  *** MOVING OR DISCARDING
 ceq ( [Ns];[Nt];[Q];[Q1];[Q] |
       < Ns @ (Q[s N ]) | MT > |
       < Nt @ (Q[ N' ]) | MT1 > | DS ) ==> = ( move # now -> [
     ( init  |
       < Ns @ (Q[  N ]) | MT >  |
       < Nt @ (Q[s N']) | MT1 > | DS) ] )
         if Q1 =/= Q .

 eq  ( [Ns];[Nt];[Q];[Q1];[Q2] | DS ) ==> =  ( move # now -> [
     ( init                    | DS ) ] ) [owise] .

 eq temp( init | DS ) = false .            *** TEMPORANEOUS STATES
 eq temp( DS ) = true [owise] .
endm
```

Fig. 2.   The transition system semantics in module CS.

the structure of a system state (CS-TYPES), some utility functions (CS-FUNCTIONS), and finally the stochastic transition system operator ==> (CS). Module CS-TYPES and module CS-FUNCTIONS are not reported for brevity. Module CS-TYPES specifies the necessary types to define the structure of a system state. In particular, sort Tuple is used to model the occurrence of a tuple in a tuple space: for instance, 'a[10] means 10 tuples of tuple type 'a occur. Sort Space is used to represent a tuple space: <0 @ ('a[10])|('b[10])|('c[10])|('d[10])> means the tuple space with identifier 0 has 10 copies of each tuple type. Module CS-FUNCTIONS defines three functions: choose takes a list of tuple type identifiers and returns one non-deterministically chosen; occurringTuples takes the content of a tuple space and returns the list of tuple types occurring in it; quantities takes the content of a tuple space and a list of tuple types and returns the cardinality of each of them.

The CS module, as depicted in Figure 2, can be viewed as the core of the Collective Sorting model. First of all, six kinds of action are defined: the former is of the kind source(0),...,source(3) and is used to start an agent working on a certain tuple space; the others are constants corresponding to the five steps of the agent agenda. The constant SS is assigned to the initial state of the system we want to simulate, where tuples are spread in different quantities in the various tuple spaces.

The stochastic transition system semantics is divided in six groups according to the actions to be executed. Initially, four actions of the first kind are allowed, each with rate 0.25. The rate of other actions is the constant now, which is assigned to a large float, meaning that these actions should happen immediately. By this modelling choice, we will simulate a system where one agent evaluates for moving a tuple at each time unit, and such an evalution is immediate. The behaviour of transitions is briefly described as follows.

a) source(i): When task init occurs in the space it is time to spawn a new agent task: any of the tuple spaces can be chosen as source, with same probability. Task [i] correspondingly replaces init, where i is the source chosen. Note that DS is a variable over DataSpace, which here matches with the rest of the system.

b) chooseTarget: To choose a target, any tuple space in 0,1,2 is tried. If the result is equal to the current source, tuple space 3 is actually taken as target. This guarantees the source and target tuple spaces to be distinct. The task moves then to state [Ns];[Nt]—source and target identifier, respectively.

c) chooseTupleType: A tuple type is chosen randomly out of those currently occurring in Ns. This is computed with functions choose and occurringTuple, and is used to avoid picking a tuple which is currently absent in the source tuple space. The task moves then to [Ns];[Nt];[QQ]—where QQ is the tuple type chosen.

d) readSource: In this step a tuple type is drawn from the source tuple space using uniform read. Expression get(QL,sample(quantities(QL, MT))) is used to sample a tuple giving higher probability to those that occur more.

e) readTarget: Similar sampling is done on the target tuple space. The task moves now to [Ns];[Nt];[Q];[Q1];[Q2], where Q1 and Q2 are the tuple types read.

f) move: If the task matches [Ns];[Nt];[Q];[Q1];[Q] and Q1 is different from Q, then a tuple of kind Q is to be moved from Ns to Nt, which is realised by properly updating the tuple counters. Otherwise ([owise]), the tuple spaces state is left unchanged. In both cases, the task gets back to init.

Finally, the temp function defines as temporary states those that do not have task init, which will then cause the simulation counter not to update.

C. Simulating the Collective Sorting

The simulation can be run by giving the MAUDE interpreter a command like

```
rewrite < [ 5000 : ( SS ) @ 0.0 ] > .
```

which executes precisely 5000 agent executions starting from state SS. Such a state is defined as a constant in the code of Figure 2, and represents a possible initial (disordered) configuration of tuples. Figure 3 shows a piece of the output produced by the execution of the simulation—where each step includes simulation countdown counter, system state, and elapsed time. After some steps, some tuple starts moving from one space to the others. After 2024 time units, for instance, tuple kind 'c is already completely collected in tuple space 2. After 4600 time units, the system converged to complete sorting, as we expected from our distributed algorithm. Chart in Figure 4 reports the dynamics of the winning tuple in each tuple space, showing e.g. that complete sorting is reached at different times in each case. The chart in Figure 5 displays instead the evolution of the tuple space 0: notice that only the tuple kind 'a aggregates here despite its initial concentration was the same of tuple kind 'b.

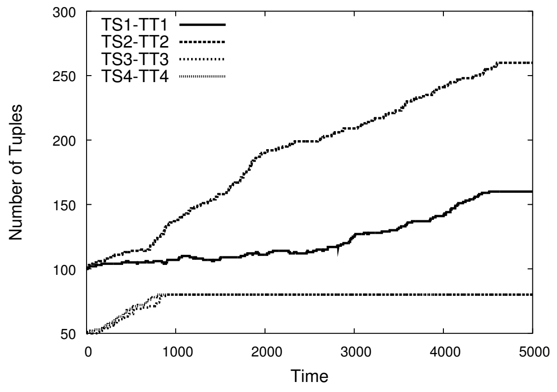


Fig. 4. Dynamics of the winning tuple in each tuple space: notice that each tuple aggregates in a different tuples space.

Although it would be possible to make some prediction, we do not know in general which tuple space will host a specific tuple kind at the end of sorting: this is an emergent property of the system and is the very result of the *interaction* of the tuple spaces through the agents! Indeed, the final result is not completely random and the concentration of tuples will evolve in the same direction *most* of the times. It is interesting to analyse the trend of the entropy of each tuple space as a way to estimate the degree of order in the system through a single value: since the strategy we simulate is trying to increase the inner order of the system we expect the entropy to decrease, as actually shown in Figure 6.

D. Adding a Load-Balancing Case

The basic strategy based on constant rates (see Section IV-B) is not very efficient, since agents are assigned to a certain tuple space also if the tuple space is already ordered! We may exploit this otherwise wasted computation by assigning idle agents to disordered tuple spaces, or rather to change the working rates of agents. This alternative therefore looks suited to realize a strategy to quicker reach the complete order of tuple spaces.

```
 <
   [5000 : init | < 0 @ ('a[100]) | ('b[100]) | ('c[10]) | ('d[10]) > |
                  < 1 @ ('a[0]) | ('b[100]) | ('c[10]) | ('d[10]) > |
                  < 2 @ ('a[10]) | ('b[50]) | ('c[50]) | ('d[10]) > |
                  < 3 @ ('a[50]) | ('b[10]) | ('c[10]) | ('d[50]) > | 'a,'b,'c,'d
         @ 0.0],
      ...
   [4000 : init | < 0 @ ('a[107]) | ('b[89]) | ('c[0]) | ('d[0]) > |
                  < 1 @ ('a[0]) | ('b[136]) | ('c[0]) | ('d[0]) > |
                  < 2 @ ('a[0]) | ('b[35]) | ('c[80]) | ('d[0]) > |
                  < 3 @ ('a[53]) | ('b[0]) | ('c[0]) | ('d[80]) > | 'a,'b,'c,'d
         @ 9.7664497212663287e+2],
   ...
   [2000 : init | < 0 @ ('a[127]) | ('b[50]) | ('c[0]) | ('d[0]) > |
                  < 1 @ ('a[0]) | ('b[210]) | ('c[0]) | ('d[0]) > |
                  < 2 @ ('a[0]) | ('b[0]) | ('c[80]) | ('d[0]) > |
                  < 3 @ ('a[33]) | ('b[0]) | ('c[0]) | ('d[80]) > | 'a,'b,'c,'d
         @ 3.0679938546387184e+3],
   ...
   [1000 : init | < 0 @ ('a[142]) | ('b[18]) | ('c[0]) | ('d[0]) > |
                  < 1 @ ('a[0]) | ('b[242]) | ('c[0]) | ('d[0]) > |
                  < 2 @ ('a[0]) | ('b[0]) | ('c[80]) | ('d[0]) > |
                  < 3 @ ('a[18]) | ('b[0]) | ('c[0]) | ('d[80]) > | 'a,'b,'c,'d
         @ 4.0271359303450395e+3],
   ...
   [438 : init  | < 0 @ ('a[160]) | ('b[0]) | ('c[0]) | ('d[0]) > |
                  < 1 @ ('a[0]) | ('b[260]) | ('c[0]) | ('d[0]) > |
                  < 2 @ ('a[0]) | ('b[0]) | ('c[80]) | ('d[0]) > |
                  < 3 @ ('a[0]) | ('b[0]) | ('c[0]) | ('d[80]) > | 'a,'b,'c,'d
         @ 4.6001450653146167e+3],
   ...
   [0 : init    | < 0 @ ('a[160]) | ('b[0]) | ('c[0]) | ('d[0]) > |
                  < 1 @ ('a[0]) | ('b[260]) | ('c[0]) | ('d[0]) > |
                  < 2 @ ('a[0]) | ('b[0]) | ('c[80]) | ('d[0]) > |
                  < 3 @ ('a[0]) | ('b[0]) | ('c[0]) | ('d[80]) > | 'a,'b,'c,'d
         @ 5.0313233386068514e+3]
>
```

Fig. 3. Result for the Collective Sorting simulation

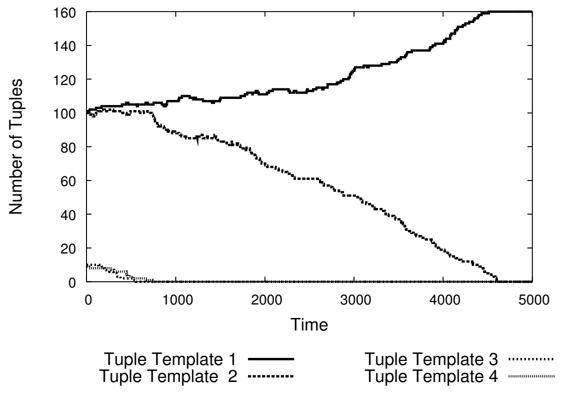


Fig. 5. Dynamic of tuple space 0: notice that only one kind of tuple aggregates here.

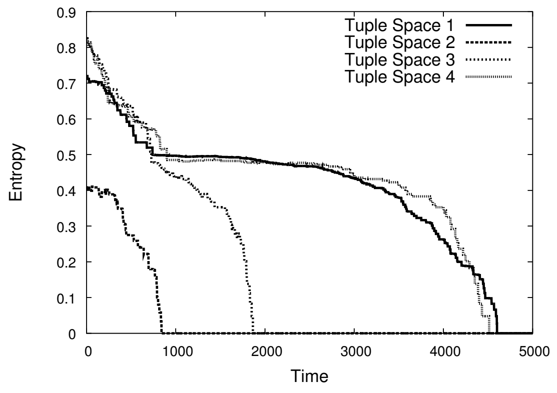


Fig. 6. Entropy of tuple spaces: they all eventually reach 0, that is, complete order.

In order to adapt the agents rate we need a measure of order: as already stated in Section III, spatial entropy may be an effective measure for system order. If we denote with q_{ij} the amount of tuples of the kind i within the tuple space j, n_j the total number of tuples within the tuple space j, and k the number of tuple kinds, then, the entropy associated with the tuple kind i within the tuple space j is

$$H_{ij} = \frac{q_{ij}}{n_j} \log_2 \frac{n_j}{q_{ij}} \tag{2}$$

and it is easy to notice that $0 \leq H_{ij} \leq \frac{1}{k} \log_2 k$. We want to express now the entropy associated with a single tuple space

$$H_j = \frac{\sum_{i=1}^{k} H_{ij}}{\log_2 k} \tag{3}$$

where the division by $\log_2 k$ is introduced in order to obtain $0 \leq H_j \leq 1$. If we have t tuple spaces then the entropy of the system is

$$H = \frac{1}{t}\sum_{j=1}^{t} H_j \tag{4}$$

where the division by t is used to normalize H, so that $0 \leq H \leq 1$. Being t the number of tuple spaces then it also represents the number of agents: let each agent work at rate $H_j r$, and tr be the maximum rate allocated to the sorting task. If we want to adapt the working rates of agents we have to scale their rate by the total system entropy, since

$$\gamma \sum_{j=1}^{t} rH_j = tr \Rightarrow \gamma = \frac{t}{\sum_{j=1}^{t} H_j} = \frac{1}{H} \tag{5}$$

hence each agent will work at rate $\frac{rH_j}{H}$ where H_j and H are computed periodically.

In order to modify the Collective Sorting model of Figure 2, we replaced the constant agents rate of the first four action (refer to Section IV-B) with the Equation 3 : hence, the activity rate of the tuple space j becomes H_j instead of `0.25`. Using *load balancing* we introduced *dynamism* in our model: indeed in each simulation step the activity rate associated with a tuple space—i.e. the probability at a given step that an agent of the tuple space is working—is no longer fixed, but it depends on the entropy of the tuple space itself. Hence, as explained above, agents belonging to completely ordered tuple spaces can consider their goal as being achieved, and hence they no longer execute tasks. Moreover, this strategy guarantees a better efficiency in the load balancing of agents work: agents working on tuple spaces with higher entropy, have a greater activity rate than the others on more ordered tuple spaces.

Using the Collective Sorting specification with variable rates, we ran the same simulation of the Section IV-C: the chart of Figure 7 shows the trend of the entropy of each tuple space. Comparing the chart with the one in Figure 6, we can observe that the entropies reach `0` faster than the case with constant rates: indeed since step `3000` every entropy within the chart in Figure 7 is `0`, while with constant rates the same result is reached only after `4600` steps. The chart in Figure 8 compares the tendency of the global entropy (see Equation 5) in the case of constant and variable rates: the trend of the two entropies represents a further proof that variable rates guarantee a faster stabilization of the system, i.e. its complete order.

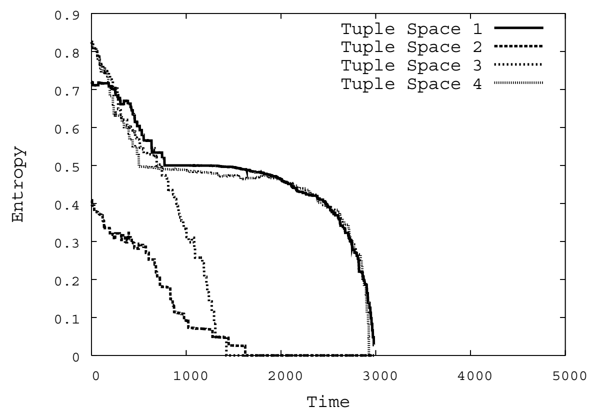


Fig. 7. Entropy of tuple spaces in the variable rate case: the system reaches the complete order since step `3000`.

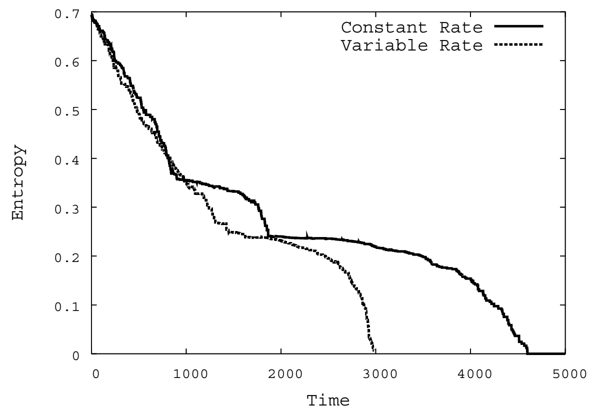


Fig. 8. Comparison of global entropy in the case of constant and variable rate: the latter reaches the complete order quicker.

V. Conclusion and Future Works

In this article we argued about the necessity of considering stochastic aspects when designing emergent coordination mechanisms: this issue is both emerging in few proposals of new coordination models and in related research contexts. We evaluated these ideas by using the MAUDE library we developed, considering and simulating a typical scenario of swarm-like coordination, the collective sorting problem, which we believe is a very paradigmatic application of emergent coordination because of its basic formulation.

Several interesting future works can be pursued:

- In the context of collective sorting, we plan to evaluate other load-balancing approaches, optimising the convergence to complete order, and working with different combinations of the number of tuple spaces and tuple kinds.
- The library itself is currently a very simple prototype, but we believe it could be improved in several ways and become a very practical simulation tool.
- Another interesting idea would be to apply our library to some existing coordination models like SwarmLinda, and provide the necessary tests for the proposed algorithms.

References

[1] E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm Intelligence: From Natural to Artificial Systems*, ser. Santa Fe Institute Studies in the Sciences of Complexity. Oxford University Press, Inc., 1999.

[2] L. Gardelli, M. Viroli, and A. Omicini, "On the role of simulations in engineering self-organising MAS: The case of an intrusion detection system in TuCSoN," in *Engineering Self-Organising Systems*, ser. LNAI, S. A. Brueckner, G. Di Marzo Serugendo, D. Hales, and F. Zambonelli, Eds. Springer, 2006, vol. 3910, pp. 153–168, 3rd International Workshop (ESOA 2005), Utrecht, The Netherlands, 26 July 2005. Revised Selected Papers.

[3] M. Mamei and F. Zambonelli, "Programming pervasive and mobile computing applications with the tota middleware," in *Pervasive Computing and Communications, 2004. PerCom 2004. Proceedings of the Second IEEE Annual Conference on*. IEEE, March 2004, pp. 263– 273.

[4] R. Menezes and R. Tolksdorf, "Adaptiveness in linda-based coordination models," in *Engineering Self-Organising Systems: Nature-Inspired Approaches to Software Engineering*, ser. LNAI, G. D. M. Serugendo, A. Karageorgos, O. F. Rana, and F. Zambonelli, Eds. Springer Berlin / Heidelberg, January 2004, vol. 2977, pp. 212–232.

[5] R. D. Nicola, D. Latella, and M. Massink, "Formal modeling and quantitative analysis of KLAIM-based mobile systems," in *SAC '05: Proceedings of the 2005 ACM symposium on Applied computing*. New York, NY, USA: ACM Press, 2005, pp. 428–435.

[6] A. Phillips, "The Stochastic Pi Machine (SPiM)," 2006, version 0.042 available online at http://www.doc.ic.ac.uk/~anp/spim/. [Online]. Available: http://www.doc.ic.ac.uk/ anp/spim/

[7] "Swarm," 2006, available online at http://www.swarm.org/.

[8] "Recursive porous agent simulation toolkit (repast)," 2006, available online at http://repast.sourceforge.net/.

[9] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott, *Maude Manual*, 2nd ed., Department of Computer Science University of Illinois at Urbana-Champaign, December 2005, version 2.2 is available online at http://maude.cs.uiuc.edu.

[10] C. Priami, "Stochastic pi-calculus," *The Computer Journal*, vol. 38, no. 7, pp. 578–589, 1995.

[11] H. Gutowitz, "Complexity-seeking ants," in *Proceedings of the Third European Conference on Artificial Life*, Deneubourg and Goss, Eds., 1993.

[12] K. Hadeli, P. Valckenaers, C. B. Zamfirescu, H. V. Brussel, B. S. Germain, T. Holvoet, and E. Steegmans, "Self-organising in multi-agent coordination and control using stigmergy," in *Engineering Self-Organising Systems: Nature-Inspired Approaches to Software Engineering*, ser. LNAI, G. D. M. Serugendo, A. Karageorgos, O. F. Rana, and F. Zambonelli, Eds. Springer Berlin / Heidelberg, January 2004, vol. 2977, pp. 105–123.

[13] A. Omicini and E. Denti, "From tuple spaces to tuple centres," *Science of Computer Programming*, vol. 41, no. 3, pp. 277–294, Nov. 2001.

[14] M. Casadei, L. Gardelli, and M. Viroli, "Simulating emergent properties of coordination in maude: the collective sorting case," in *5th International Workshop on the Foundations of Coordination Languages and Software Architectures, Bonn, Germany*, August 2006, to appear into.

Minority Game: A Logic-Based Approach in TuCSoN

Enrico Oliva Mirko Viroli Andrea Omicini
ALMA MATER STUDIORUM—Università di Bologna
via Venezia 52, 47023 Cesena, Italy
E-mail:{enrico.oliva,mirko.viroli,andrea.omicini}@unibo.it

Abstract— **Minority Game is receiving an increasing interest because it models emergent properties of complex systems including rational entities, such as for instance the evolution of financial markets. As such, Minority Game provides for a simple yet stimulating scenario for system simulation.**

In this paper, we aim at presenting a logic approach to the Minority Game whose goal is to overcome the well-known limits of the equation model in the verification of the system behaviour. We realise the social system simulation using a novel MAS meta-model based on agents and artifacts, where the agent rationality is obtained using a BDI architecture.

To this end, we adopt the TuCSoN infrastructure for agent coordination, and its logic-based tuple centre abstractions as artifact representatives. By implementing Minority Game over TuCSoN, we show some of the benefits of the artifact model in terms of flexibility and controllability of the simulation.

A number of parameters can affect the behaviour of Minority Game simulation: such parameters are explicitly represented in the coordination artifact, so that they can be tuned up during the simulation. In particular, experiments are shown where memory size and number of wrong moves are adopted as the tuning parameters.

I. Introduction

Minority Game (MG) is a mathematical model that takes inspiration from the “El Farol Bar” problem introduced by Brian Arthur (1). It is based on a simple scenario where at each step a set of agents perform a boolean vote which conceptually splits them in two classes: the agents in the smaller class win. In this game, a rational agent keeps track of previous votes and victories, and has the goal of winning throughout the steps of the game—for which a rational strategy has to be figured out. Several researches showed that, although very simple, this model takes into account crucial aspects of some interesting complex systems coupling rationality with emergence: e.g. bounded rationality, heterogeneity, competition for limited resources, and so on. For instance, MG is a good model to study market fluctuation, as an emergent property resulting from interactions propagating from micro scale (agent interaction) to macro scale (collective behaviour).

As showed by (2), a multiagent system (MAS) can be used to realise a MG simulation—there, BDI agents provide for rationality and planning. An agent-based simulation is particularly useful when the simulated systems include autonomous entities that are diverse, thus making it difficult to exploit the traditional framework of mathematical equations.

The Minority Game is a social simulation that aims at reproducing a simplified human social scenario. A (human) society is composed by different kinds of people with different behaviours, and its composition affects the progress of the game. In principle, a logic-based approach based on BDI agent makes it easier to explicitly model a variety of diverse social behaviours. Also, in this scenario, argumentation theory (3) is useful to model the information exchange and sharing between humans/agents so as to improve the agent reasoning abilities, as well as to provide a more realistic simulation of a society.

In this paper we proceed along this direction, and adopt a novel MAS meta-model based on the notion of artifact (4). The notion of artifact is inspired by Activity Theory (5): it represents those abstractions living in the MAS environment that provide a function, which agents can exploit to achieve individual and social goals. The engineering principles promoted by this meta-model makes it possible to flexibly balance the computational burden of the whole system between autonomy of the agents and the designed behaviour of artifacts.

In order to implement MG simulations we adopt the TuCSoN infrastructure for agent coordination (6), which introduces tuple centres as artifact representatives. A tuple centre is a programmable coordination medium living in the MAS environment, used by agents interacting by exchanging tuples (logic tuples in the case of TuCSoN logic tuple centres). As we are not concerned much with the mere issues of agent intelligence, we rely here on a weak form of rationality, through logic-based agents adopting pre-compiled plans called *operating instructions* (7).

By implementing MG over TuCSoN, we can experiment with flexibility and controllability of the artifact model, and see if and how they apply to the simulation – in particular, artifacts allow for a greater level of controllability with respect to agents. To this end, in this paper we show how the model allows some coordination parameters to be changed during the run of a simulation with no need to stop the agents: this can be useful e.g. to change the point of equilibrium, controlling the collective behaviour resulting by interactions propagated from the entities at the micro level.

The remainder of this paper is organised as follows. First, we introduce the general simulation framework based on agents and artifacts. Then, we provide the reader with some relevant details of the Minority Game. Some quantitative results of MG simulation focussing on system dynamics and run-time changes are presented, just before final remarks.

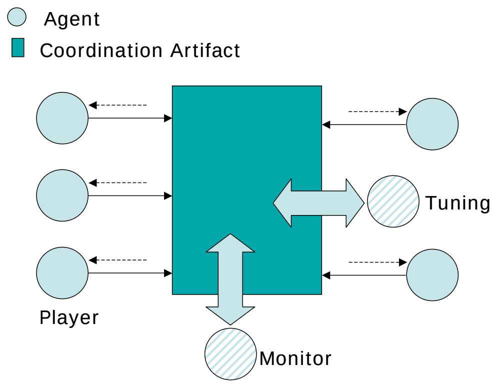


Fig. 1. TuCSoN Simulation Framework for MG

II. The TuCSoN Framework for Simulation

The architecture proposed for MAS simulation is based on TuCSoN (6), which is an infrastructure for the coordination of MASs. TuCSoN provides agents with an environment made of logic tuple centres, which are logic-based programmable tuple spaces. The language used to program the coordination behaviour of tuple centres is ReSpecT, which specifies how a tuple centre has to react to an observable event (e.g. when a new tuple is inserted) and has to accordingly change the tuple-set state (8). Tuple centres are a possible incarnation of the coordination artifact notion (9), representing a device that persists independently of agent life-cycle and provides services to let agents participate to social activities.

In our simulation framework we adopt logic-based agents, namely, agents built using a logic programming style, keeping a knowledge base (KB) of facts and acting according to some rule—rules and facts thus forming a logic theory. The implementation is based on tuProlog technology[1] for Java-Prolog integration, and relies on its inference capabilities for agent rationality. Agents roughly follow the BDI architecture (as showed in Figure 2), as the KB models agent beliefs while rules model agent intentions.

To coordinate agents we take inspiration from natural systems like ant-colonies, where coordination is achieved through the mediation of the environment: our objective is to have a possibly large and dynamic set of agents which coordinate each other through the environment while bringing about their goals.

Externally, we can observe overall system parameters by inspecting the environment, namely, the tuple centres agents interact with. In this way we can try different system behaviours changing only the coordination behaviour of the environment. Furthermore we can change, during the simulation, some coordination parameters (expressed as tuples in a tuple centre), programming and then observing the transition of the whole system either to a new point of equilibrium or to a divergence.

Three kinds of agents are used in our simulation: player agents, monitor agents and tuning agents (as depicted in Figure 1): all the agents share the same coordination artifact. The agent types differ because of their role and behaviour: player agents play MG, the monitor agent is an observer of interactions which visualises the progress of the system, the tuning agent can change some rules or parameters of coordination, and drives the simulation to new states. Note that the main advantage of allowing a dynamic tuning of parameters instead of running different simulations lays in the possibility of tackling emergent aspects which would not necessarily appear in new runs.

The main control loop of a player agent is a sequence of actions: observing the world (perception), updating its KB (effects), scheduling next intention (precondition), elaborating and executing a plan (action). This structure is depicted in Figure 2. Moreover, in order to connect agent mental states with interactions, we use the concept of action preconditions and perception effects as usual.

[1]`http://tuprolog.alice.unibo.it`

III. Minority Game

MG was introduced and first studied by (10), as a means to evaluate a simple model where agents compete through adaptation for finite resources. MG is a mathematical representation from 'El Farol Bar' problem introduced by (1), providing an example of inductive reasoning in scenarios of bounded rationality. The game consists in an odd number N of agents: at each discrete time step t of the game an agent i takes an action $a_i(t)$, either 1 or -1. Agents taking the minority action win, whereas the majority looses. After a round, the total action result is calculated as:

$$A(t) = \sum_{i}^{N} a_i(t)$$

In order to take decisions agents adopt strategies. A strategy is a choosing device that takes as input the last m winning results, and provides the action (1 or -1) to perform in the next time step. The parameter m is the size of the memory of the past results (in bits), and 2^m is therefore the potential past history that defines the number of possible entries for a strategy.

The typical strategy implementation is as follows. Each agent carries a sequence of 2^m actions, called a strategy, e.g.

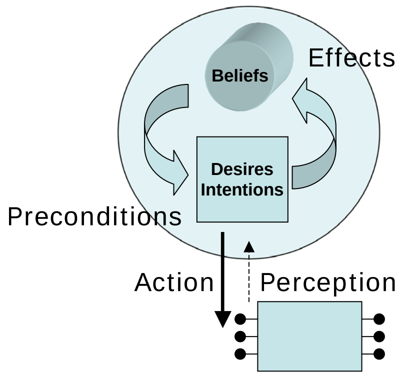


Fig. 2. Agent Architecture

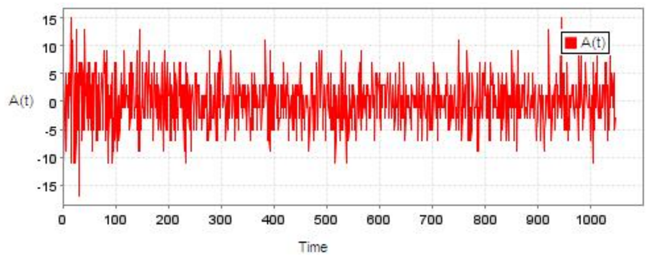


Fig. 3. Typical Time evolution of the Original MG with $N = 51$, $m = 5$ and $s = 2$

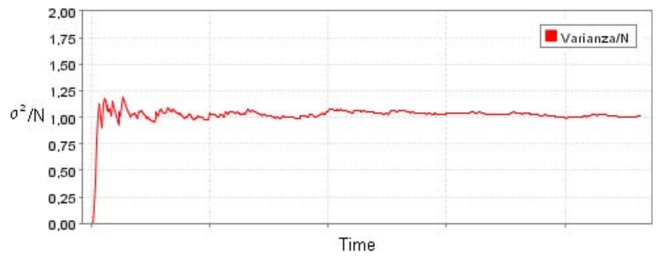


Fig. 4. Variance of the Game with 11 Random Agents

$m = 3\ 2^3 actions = [+1, +1, -1, -1, +1, -1, +1, +1]$. The information on past m wins is stored considering the success of $-$ group if $A(t) > 0$ or $+$ group if $A(t) < 0$. Such a past history is mapped on the natural number that results by considering $-$ as 0 and $+$ as 1. Such a number is used as position in the sequence of the next action to take: for instance, if $[-, +, -]$ is the past winning group, we read it as 010 (that is, 2), and accordingly pick the decision in position 2 inside $[+1, +1, -1, -1, +1, -1, +1, +1]$, that is -1.

Each agent actually carries a number $s \geq 2$ of strategies. During the game the agent evaluates all its strategies according to their success, and hence at each step it decides based on the most successfull strategy so far. Figure 3 shows a typical evolution of the game.

One of the most important applications of MG is in the market models: (11) use MG as a coarse-grained model for financial markets to study their fluctuation phenomena and statistical properties. Even though the model is coarse-grained and provides an over-simplified micro-scale description, it anyway captures the most relevant features of system interaction, and generates collective properties that are quite similar to those of the real system.

Another point of view, presented e.g. by (12), considers the MG as a point in space of a Resource Allocation Game (RAG). In this work a generalisation of MG is presented that relaxes the constraints on the number of resources, studying how the system behaves within a given range.

A. MG Logic-Based Approach

MG can be considered a social simulation that aims to reproduce a simplified human scenario. Each (human) agent, in this scenario, must do a choice under the minority global rule. In order to study the system composed by different kinds of players with different behaviours, we here adopt a logic-based approach to build the players. In this way, it is possible to observe particular social behaviours which would otherwise remain hidden in the approximation of the mathematical model.

A more recent paper (2) observes that MG players could be naturally modelled as agents with a full BDI model, and adopts a new adaptive stochastic MG with dynamically evolving strategies in the simulation. We can then apply our simulation framework, with Logic Agents and Coordination Artifacts, to test the MG from a logic-based point of view, and to experiment with some dynamic tuning strategy.

The next step is to consider players as in an argumentation scenario (3), where agents have the ability to exchange arguments with the purpose to make their own choice or to persuade others to change theirs.

B. MG Performance

In order to track the performance of an MG system, the most interesting quantity is *variance*, defined as $\sigma^2 = \overline{[A(t) - \overline{A(t)}]^2}$: it shows the variability of the bets around the average value $\overline{A(t)}$. In particular, the normalised version of variance $\rho = \sigma^2/N$ is considered.

Generally speaking, variance is the inverse of global efficiency: as variance decreases agent coordination improves, making more agents winning. Variance is interestingly affected by the parameters of the model, such as number of agents (N), memory (m) and number of strategies (s): in particular, the fluctuation of variance is shown to depend only on the ratio $\alpha = 2^m/N$ between agent memory and the number N of agents.

For large values of α—the number of agents is small with respect to the number of possible histories—the outcomes are seemingly random: the reason for this is that the information that agents observe about the past history is too complex for their limited processing analysis.

When new agents are added, fluctuation decreases and agents perform better by choosing randomly, in this case $\rho = 1$ and $\alpha \approx 1/2$, as visible in the results of our simulation in Figure 4—the game enters into a regime where the loosing group is close to $N/2$, hence we might say coordination is performing well.

If the number of agents increase further, fluctuations rapidly increase beyond the level of random agents and the game enters into the crowded regime. With a low value of α the value of σ^2/N is very large: it scales like $\sigma^2/N \approx \alpha^{-1}$.

The results of other observations suggest that the behaviour of MG can be classified in two phases: an information-rich *asymmetric* phase, and an unpredictable or *symmetric* phase. A phase transition is located where σ^2/N attains its minimum ($\alpha_c = 1/2$), and it separates the symmetric phase with $\alpha < \alpha_c$ from an asymmetric phase with $\alpha > \alpha_c$.

All these cases have been observed with the TuCSoN simulation framework described in next section.

IV. The Simulation Framework

The construction of MG simulations with MASs is based on the TuCSoN framework and on tuProlog as an inferential engine to program logic agents. The main innovative aspect of this MG simulation is the possibility of studying the evolution of the system with particular and different kinds of agent behaviour at the micro level, imposed as coordination parameters which are changed on-the-fly.

A. Operating Instructions

Each agent has an internal plan, structured as an algebraic composition of allowed actions (with their preconditions) and perceptions (with their effects), that enables the agent to use the coordination artifact to play the MG. This plan can be seen as Operating Instructions (7), a formal description based on Labelled Transition Systems (LTS) that the agent reads to understand what its step-by-step behaviour should be. Through an inference process, the agent accordingly chooses the next action to execute, thus performing the cycle described in Section II.

Operating instructions are expressed by the following theory:

```
% pre=Preconditions
% eff=Effects
% act=Action
% per=Perception
firststate(agent(first,[])).
definitions([
  def(first,[],...),
  %definition of the main control loop
  def(main,[S],
       [act(out(play(X)),pre(choice(S,X))),
        per(in(result(Y)),eff(res(Y))),
        agent(main,[S])]
  ),
  ...
]).
```

The first part of operating instructions is expressed by term `first`, where the agent reads the game parameters that are stored in the KB, and randomly creates its own set of strategies.

In the successive part `main`, the agent executes its main cycle. It first puts tuple `play(X)` in the tuple space, where $X = \pm 1$ is agent vote. The precondition of this action `choice(S,X)` is used to bind in the KB X with the value currently chosen by the agent according to strategy S. Then, the agent gets the whole result of the game in tuple `result(Y)` and applies it to its KB. After this perception, the cycle is iterated again.

B. Tuple Centre Behaviour

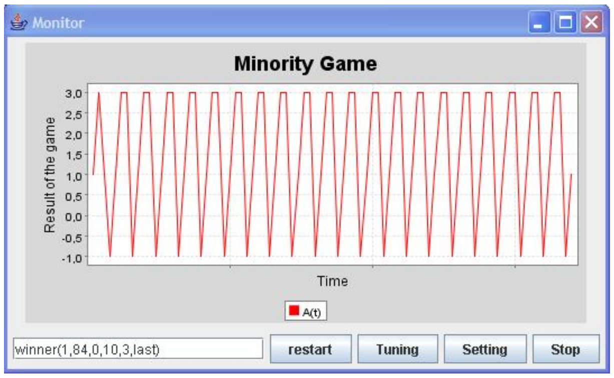
The interaction protocol between agents and the coordination artifact is then simply structured as follows. First each agent puts the tuple for its vote. When the tuples for all agents have been received, the tuple centre checks them, computes the result of the game—either 1 or -1 is winning—and prepares a result tuple to be read by agents.

The ReSpecT program for this behaviour is loaded in the tuple centre by a configuration agent at bootstrap, through operation `set_spec()`. The following ReSpecT reaction is fired when an agent inserts tuple `play(X)`, and triggers the whole behaviour:

```
reaction(out(play(X)),(
  %read the last value of count
  in_r(count(Y)),
  Z is Y+1,
  %calculate the partial result
  in_r(sum(M)),
  V is M+X,
  out_r(sum(V)),
  %store the new value of count
  out_r(count(Z))
  %this action will be catch
)).
```

This reaction considers the bet (`X`), counts the bets (`Z`), and computes the partial result of the game (`V`). When all the agents have played, the artifact produces the tuple `winner(Result,Turn,NumberOfLoss,MemorySize,last/more)` which is the main tuple of MG coordination.

```
reaction(out_r(count(X)),(
  %check if all agents have already played
  rd_r(numag(Num)),
  X=:=Num,
  in_r(totcount(T)),
  Turn is T+1,
  rd_r(game(G)),
  %read the result of the game
  in_r(sum(Result)),
  %reset the sum value
  out_r(sum(0)),
  rd_r(countsession(CS)),
  in_r(count(Y)),
  %reset the count value
  out_r(count(0)),
  %calculate variance
  in_r(qsum(SQ)),
  NSQ is Result*Result+SQ,
  out_r(qsum(NSQ)),
  %calculate mean
  in_r(totsum(R)),
  NewS is R+Result,
  out_r(totsum(NewS)),
  rd_r(numloss(NumberOfLoss)),
  rd_r(mem(MemorySize)),
  % put out the tuple with the result
  out_r(winner(Result,Turn,NumberOfLoss,
  MemorySize,G)),
  out_r(totcount(Turn))
)).
```



```
winner(1,84,0,10,3,last)
```

Fig. 5. Interface of the Monitor Agent

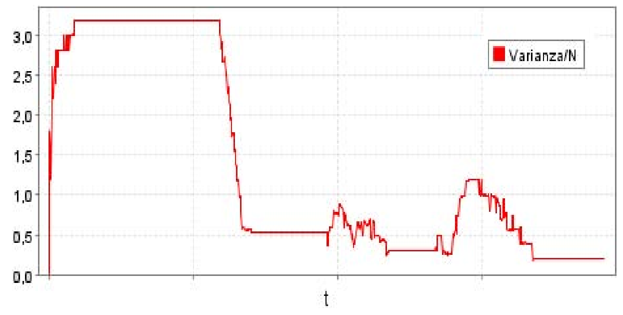


Fig. 6. Variance of the System with Initial Parameters $N = 5$ and $m = 3$

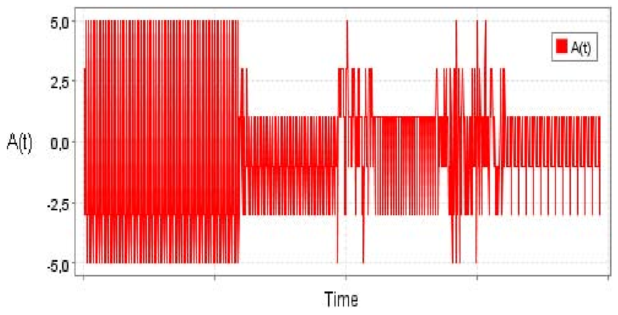


Fig. 7. System Evolution of the Variance in Figure 6

The `winner` tuple contains the result of the game (`Result`), the number of steps (`Turn`), two tuning parameters (`NumberOfLoss` and `MemorySize`) and one constant to communicate agents whether they have to stop or to play further (`last/more`). Figure 5 reports the graphical interface of the monitor agent that during its life-time reads the tuple `winner` and draws variance.

C. Tuning the Simulation

In classical MG simulation there are a number of parameters that can affect the system behaviour, which are explicitly represented in the tuple centre in form of tuples: the number of agents `numag(X)`, memory size `mem(X)`, and the number of strategies `numstr(X)`. In our framework, we have introduced as a further parameter the number of wrong moves after which the single agent should be recalculate own strategy, represented as a tuple `numloss(X)`. Such a threshold is seemingly useful to break the symmetry in the strategy space when the system is in a pathological state, i.e., when all agents have the same behaviour and the game oscillates from minimum to maximum value.

In our framework, it is possible to explore the possibility to dynamically tune up the coordination rules by changing `numloss` and `mem` coordination parameters, which are stored as tuples in the coordination artifact. The simulation architecture built in this way, in fact, allows for on-the-fly change of some game configuration parameters—such as the dimension of agent memory—with no need to stop the simulation and re-program the agents.

By changing the parameters, the tuning agent can drive the system from an equilibrium state to another, by controlling agent strategies, the dimension of memory, or the number of losses that an agent can accept before discarding a strategy. This agent observes system variance, and decides whether and how to change tuning parameters: reference variance is calculated by first making agents playing the game randomly—see Figure 4. The new value of parameters is stored in tuple centre through tuples `numloss(NumberOfLoss)` and `mem(MemorySize)`, the rules of coordination react and update the information that will be read by the agents.

D. Simulation Results

The result of the tuned simulation in Figures 6 and 7 shows how the system changes its equilibrium state and achieves a better value of variance.[2] In this simulation the tuning agent is played by a human that observes the evolution of the system and acts through the tuning interface to change the coordination parameters, such as threshold of losses and memory, hopefully finding new and better configurations. The introduction of the threshold of losses in the agent behaviour is useful when the game is played by few agents: these parameters enable system evolution and a better agent cooperative behaviour.

V. Conclusion

In this paper, we aim at introducing new perspectives on agent-based simulation by adopting a novel MAS meta-model based on agents and artifacts, and by applying it to Minority Game simulation. We implement and study MG over the TuCSoN coordination infrastructure, and show some benefits of the artifact model in terms of flexibility and controllability of the simulation. In particular, in this work we focus on the possibility to build a feedback loop on the rules of coordination driving a system to a new and better equilibrium state. Many related agent simulation tools actually exist: as this paper is a starting point, we plan to perform a systematic comparison of their expressiveness and features. In the future, we are interested in constructing an intelligent and adaptive tuning agent with a BDI architecture, substituting the human agent in driving the evolution over time of the system behaviour.

VI. Acknowledgements

The first author of this paper, Enrico Oliva, would like to warmly thank Dr. Peter McBurney and the Department of Computer Science at University of Liverpool for their scientific support and their hospitality during his stay in Liverpool, when this paper was mostly written.

[2]In Figure 6, the first phase of equilibrium is followed by a second one obtained by changing the threshold parameter $S = 5$. Finally, a third phase is obtained changing the dimension of the memory to $m = 5$.

References

[1] W. B. Arthur, "Inductive reasoning and bounded rationality (the El Farol problem)," *American Economic Review*, vol. 84, no. 2, pp. 406–411, May 1994.

[2] W. Renz and J. Sudeikat, "Modeling Minority Games with BDI agents – a case study," in *Multiagent System Technologies*, ser. LNCS, T. Eymann, F. Klügl, W. Lamersdorf, M. Klusch, and M. N. Huhns, Eds. Springer, 2005, vol. 3550, pp. 71–81, 3rd German Conference (MATES 2005), Koblenz, Germany, 11-13 Sept. 2005. Proceedings. [Online]. Available: http://www.springerlink.com/link.asp?id=y62q174g56788gh8

[3] S. Parsons and P. McBurney, "Argumentation-based communication between agents." in *Communication in Multiagent Systems*, ser. Lecture Notes in Computer Science, M.-P. Huget, Ed., vol. 2650. Springer, 2003, pp. 164–178.

[4] A. Ricci, M. Viroli, and A. Omicini, "Programming MAS with artifacts," in *Programming Multi-Agent Systems*, ser. LNAI, R. P. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni, Eds. Springer, Mar. 2006, vol. 3862, pp. 206–221, 3rd International Workshop (PROMAS 2005), AAMAS 2005, Utrecht, The Netherlands, 26 July 2005. Revised and Invited Papers. [Online]. Available: http://www.springerlink.com/openurl.asp?genre=article&issn=0302-9743&volume=3862&spage=206

[5] A. Ricci, A. Omicini, and E. Denti, "Activity Theory as a framework for MAS coordination," in *Engineering Societies in the Agents World III*, ser. LNCS, P. Petta, R. Tolksdorf, and F. Zambonelli, Eds. Springer-Verlag, Apr. 2003, vol. 2577, pp. 96–110.

[6] A. Omicini and F. Zambonelli, "Coordination for Internet application development," *Autonomous Agents and Multi-Agent Systems*, vol. 2, no. 3, pp. 251–269, Sept. 1999.

[7] M. Viroli and A. Ricci, "Instructions-based semantics of agent mediated interaction," in *3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*, N. R. Jennings, C. Sierra, L. Sonenberg, and M. Tambe, Eds., vol. 1. New York, USA: ACM, 19–23 July 2004, pp. 102–109. [Online]. Available: http://portal.acm.org/citation.cfm?id=1018409.1018737

[8] A. Omicini and E. Denti, "Formal ReSpecT," *Electronic Notes in Theoretical Computer Science*, vol. 48, pp. 179–196, June 2001.

[9] A. Omicini, A. Ricci, M. Viroli, C. Castelfranchi, and L. Tummolini, "Coordination artifacts: Environment-based coordination for intelligent agents," in *3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*, N. R. Jennings, C. Sierra, L. Sonenberg, and M. Tambe, Eds., vol. 1. New York, USA: ACM, 19–23 July 2004, pp. 286–293. [Online]. Available: http://portal.acm.org/citation.cfm?id=1018409.1018752

[10] D. Challet and Y.-C. Zhang, "Emergence of cooperation and organization in an evolutionary game," *Physica A: Statistical and Theoretical Physics*, vol. 246, no. 3–4, pp. 407–418, Dec. 1997. [Online]. Available: http://dx.doi.org/10.1016/S0378-4371(97)00419-6

[11] D. Challet, M. Marsili, and Y.-C. Zhang, "Modeling market mechanism with minority game," *Physica A: Statistical and Theoretical Physics*, vol. 276, no. 1–2, pp. 284–315, Feb. 2000. [Online]. Available: http://dx.doi.org/10.1016/S0378-4371(99)00446-X

[12] H. V. D. Parunak, S. Brueckner, J. Sauter, and R. Savit, "Effort profiles in multi-agent resource allocation," in *1st International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2002)*, C. Castelfranchi and W. L. Johnson, Eds. Bologna, Italy: ACM, 15–19 July 2002, pp. 248–255.

[13] N. R. Jennings, C. Sierra, L. Sonenberg, and M. Tambe, Eds., *3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*. New York, USA: ACM, 19–23 July 2004.

Reasoning about Goals in BDI Agents: the PRACTIONIST Framework

Vito Morreale*, Susanna Bonura*, Giuseppe Francaviglia*, Fabio Centineo*,
Massimo Cossentino†§, and Salvatore Gaglio†‡
*R&D Laboratory - ENGINEERING Ingegneria Informatica S.p.A.
†ICAR-Italian National Research Council
‡DINFO-University of Palermo
§SET - Universit de Technologie Belfort-Montbliard, France

***Abstract*— The representation of goals and the ability to reason about them play an important role in goal-oriented requirements analysis and modelling techniques, especially in agent-oriented software engineering. Moreover goals are more useful and stable abstractions than others (e.g. user stories) in the analysis and design of software applications. Thus, the PRACTIONIST framework supports a goal-oriented approach for developing agent systems according to the Belief-Desire-Intention (BDI) model.**

In this paper we describe the goal model of PRACTIONIST agents, in terms of the general structure and the relations among goals. Furthermore we show how PRACTIONIST agents use their goal model to reason about goals during their deliberation process and means-ends reasoning as well as while performing their activities.

I. Introduction

With the increasing management complexity and maintenance cost of advanced information systems, attention in recent years has fallen on self-* systems and particularly on the autonomic computing approach and autonomic systems. In [1] authors argue that adopting a design approach that supports the definition of a space of possible behaviours related to the same function is one of the ways to make a system autonomic. Then the system should be able to select at runtime the best behaviour on the basis of the current situation. Goals can be used as an abstraction to model the functions around which the systems can autonomously select the proper behaviour.

In this view, the explicit representation of goals and the ability to reason about them play an important role in several requirements analysis and modelling techniques, especially when adopting the agent-oriented paradigm.

In this area, one of the most popular and successful agent models is the BDI [2], which derives from the philosophical tradition of practical reasoning first developed by Bratman [3]. It states that agents decide, moment by moment, which actions to perform in order to pursue their goals. Practical reasoning involves a deliberation process, to decide what states of affairs to achieve, and a means-ends reasoning, to decide how to achieve them.

Nevertheless there is a gap between BDI theories and several implementation [4]. Indeed, most of existing BDI agent platforms (e.g. JACK [5], JAM [6]) generally use goals instead of desires. Moreover, the actual implementations of mental states differ somewhat from their original semantics: desires (or goals) are treated as event types (such as in AgentSpeak(L) [7]) or procedures (such as in 3APL [8]) and intentions are executing plans. Therefore the deliberation process and means-ends reasoning are not well separated, as being committed to an intention (ends) is the same as executing a plan (means).

Moreover, some available BDI agent platforms do not support the explicit representation and implementation of goals or desires with their properties and relations, but they deal with them in a procedural and event-based fashion. As a result, while such an explicit representation of goals provide useful and stable abstractions when analysing and designing agent-based systems, there is a gap between the products of those phases and what development frameworks support.

According to Winikoff et al. [4], ”by omitting the declarative aspect of goals the ability to reason about goals is lost”. What is actually lost is the ability to *know* if goals are impossible, achieved, incompatible with other goals, and so forth. This in turn can support the *commitment strategies* of agents and their ability to autonomously drop, reconsider, replace or pursue goals.

However, some other BDI agent platforms deal with declarative goals. Indeed, in JADEX goals are explicitly represented according to a generic model, enabling the agents to handle their life cycle and reasoning about them [9]. Nevertheless, the model defined in JADEX does not deal with relations among goals.

The PRACTIONIST framework [10] adopts a goal-oriented approach to develop BDI agents and stresses the separation between the deliberation process and the means-ends reasoning, with the abstraction of goal used to formally define both desires and intentions during the deliberation phase. Indeed, in PRACTIONIST a goal is considered as an analysis, design, and implementation abstraction compliant to the semantics described in this paper. In other words, PRACTIONIST agents can be programmed in terms of goals, which then will be related to either desires or intentions according to whether some specific conditions are satisfied or not.

After a brief overview of the general structure of PRACTIONIST agents and their execution model (section II), this paper addresses the definition of the goal model (section III). We also describe how PRACTIONIST agents are able to reason about available goals according to their goal model, current beliefs, desires, and intentions (see section IV). All aforementioned issues and the proposed model are fully implemented in the PRACTIONIST framework and available when developing applications by using the goal-oriented approach and the concepts described in this paper (section V). Finally, in section VI we present a simple example that illustrates the definition and the usage of goals and their relations.

II. PRACTIONIST AGENTS

The PRACTIONIST framework aims at supporting the programmer in developing BDI agents and is built on top of JADE [11], a widespread platform that implements the FIPA[1] specifications. Therefore, our agents are deployed within JADE containers and their main cycle is implemented by means of a JADE cyclic behaviour.

A PRACTIONIST agent is a software component endowed with the following elements:

- a set of *perceptions* and the corresponding *perceptors* that listen to some relevant external stimuli;
- a set of *beliefs* representing the information the agent has got about both its internal state and the external environment;
- a set of *goals* the agent wishes or wants to pursue. They represent some states of affairs to bring about or activities to perform and will be related to either its desires or intentions (see below);
- a set of *goal relations* the agent uses during the deliberation process and means-ends reasoning;
- a set of *plans* that are the means to achieve its intentions;
- a set of *actions* the agent can perform to act over its environment; and
- a set of *effectors* that actually execute the actions.

Beliefs, plans, and the execution model are briefly described in this section, while goals are the subject of this paper and are presented in the following sections. However, for a detailed description of the structure of PRACTIONIST agents, the reader should refer to [10].

The BDI model refers to beliefs instead of knowledge, as beliefs are not necessarily true, while *knowledge* usually refers to something that is true [12]. According to this, an agent may believe true something that is false from the other agents' or the designer's point of view, but the idea is just to provide the agents with a subjective window over the world.

Therefore each PRACTIONIST agent is endowed with a prolog belief base, where beliefs are asserted, removed, or entailed through inference on the basis of KD45 modal logic rules [12] and user-defined formulas. Currently the PRACTIONIST framework supports two prolog engines, i.e. SWI-Prolog[2] and one that was derived from TuProlog[3].

In the PRACTIONIST framework plans represent an important container in which developers define the actual behaviors of agents.

[1]http://www.fipa.org
[2]http://www.swi-prolog.org
[3]http://tuprolog.alice.unibo.it

Each agent may own a declared set of plans (the *plan library*), each specifying the course of acts the agent will undertake in order to pursue its intentions, or to handle incoming perceptions, or to react to changes of its beliefs.

PRACTIONIST plans have a set of slots that are used by agents during the means-ends reasoning and the actual execution of agent activities. Some of these slots are: the trigger event, which defines the event (i.e. goals, perceptions, and belief updating) each plan is supposed to handle; the context, a set of condition that must hold before the plan can be actually performed; the body, which includes the acts the agent performs during the execution of the plan.

Through their perceptors, agents search for stimuli (perceptions) from the environment and transform them into (external) *events*, which in turn are put into the *Event Queue* (figure 1). Such a queue also contains internal events, which are generated when either an agent is committed to a goal or there is some belief updates. The former type of internal events is particularly important in PRACTIONIST agents, as described in the following sections.

The main cycle of a PRACTIONIST agent is implemented within a cyclic behaviour, which consists of the following steps.

1) it selects and extracts an event from the queue, according to a proper *Event Selection* logic;
2) it handles the selected event through the following *means-ends reasoning* process: (i) the agent figures out the *practical* plans, which are those plans whose trigger event matches the selected event (*Options* in figure 1); (ii) among practical plans, the agent detects the *applicable* ones, which are those plan whose context is believed true, and selects one of them (*main plan*); (iii) it builds the *intended means*, which will contain the main plan and other alternative practical plans. In case of goal event updates the corresponding intended means stack; otherwise it creates a new intended means stack.

It should be noted that every intended means stack can contain several intended means, each able to handle a given event, possibly through several alternative means.

Moreover all intended means stacks are concurrently executed, in order to provide the agents with the capability of performing several activities (perhaps referring to related or non-related objectives) in parallel. When executing each stack, the top level intended means is in turn executed, by performing its main plan. If it fails for some reason, one of alternative plans is then performed, until the corresponding ends (related to the triggering event) is achieved.

During the execution of a plan, several acts can be performed, such as *desiring* to bring about some states of affairs or to perform some action, *adding* or *removing* beliefs, *sending* ACL messages, and so forth. Particularly, desiring to pursue a goal triggers a deliberation/filtering process, in which the agent figures out whether that goal must be actually pursued or not, on the basis of the goal model declared for that agent.

The interaction among intended means belonging to different stacks can occur at a goal level, since each plan could wait

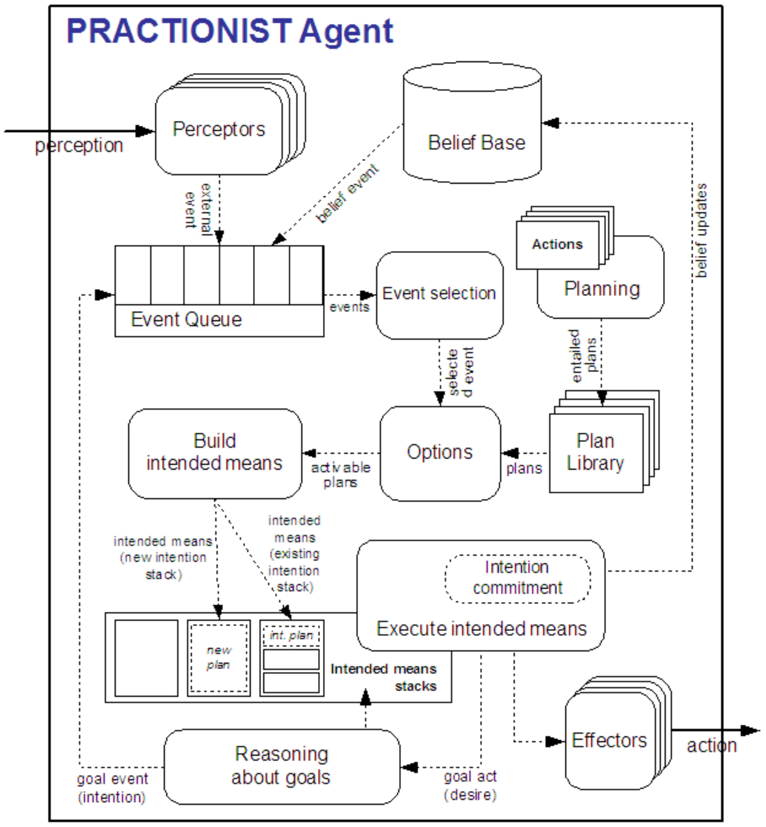


Fig. 1. PRACTIONIST Agent Architecture

for the success/failure of some goal that the agent is pursuing through another intended means.

III. Goal Model

In the PRACTIONIST framework, a goal is an objective to pursue and we use it as a mean to transform desires into intentions through the satisfaction of some properties. In other words, our agents are programmed in terms of goals, which then will be related to either desires or intentions according to whether some specific conditions are satisfied or not.

Formally, a PRACTIONIST *goal* g is defined as follows:

$$g = \langle \sigma_g,\ \pi_g \rangle \qquad (1)$$

where:

- σ_g is the *success condition* of the goal g;
- π_g is the *possibility condition* of the goal g stating whether g can be achieved or not.

Since we consider such elements as local properties of goals, in the PRACTIONIST framework we defined them as operations that have to be implemented for each kind of goal (figure 3).

In order to describe the goal model, we first provide some definitions about the properties of goals.

Definition 1 A goal g_1 is *inconsistent* with a goal g_2 ($g_1 \perp g_2$) if and only if when g_1 succeeds, then g_2 fails.

Definition 2 A goal g_1 *entails* a goal g_2 or equivalently g_2 is *entailed by* g_1 ($g_1 \longrightarrow g_2$) if and only if when g_1 succeeds, then also g_2 succeeds.

Definition 3 A goal g_1 is a *precondition* of a goal g_2 ($g_1 \mapsto g_2$) if and only if g_1 must succeed in order to be possible to pursue g_2.

Definition 4 A goal g_1 *depends* on a goal g_2 ($g_1 \hookrightarrow g_2$) if and only if g_2 is precondition of g_1 and g_2 must be successful while pursuing g_1.

Therefore the dependence is a stronger form of precondition. Both definitions let us specify that some goals must be successful before (and during, in case of dependency) pursuing some other goals (refer to section IV for more details).

Now, given a set G of goals and based on the above definitions, it is also possible to define some relations between those goals.

Definition 5 The inconsistency $\Gamma \subseteq G \times G$ is a binary symmetric relation on G, defining goals that are inconsistent with each other. Formally,

$$\Gamma = \{(g_i, g_j) \quad i,j = 1, ..., |G| \quad : \ g_i \perp g_j\}\,. \qquad (2)$$

When two goals are inconsistent with each other, it might be useful to specify that one is preferred to the other. We denote that g_i is preferred to g_j with $g_i \succ g_j$.

Definition 6 The relation of preference $\Gamma^{'} \subseteq \Gamma$ defines the pair of goals (g_i, g_j) where $g_i \perp g_j$ and $g_i \succ g_j$. Formally,

$$\Gamma' = \{(g_i, g_j) \in \Gamma : g_i \succ g_j\} . \tag{3}$$

Therefore if there is no preference between two inconsistent goals, the corresponding pair does not belong to the set $\Gamma^{'}$. Moreover, since several goals can be pursued in parallel, there is no need to prefer some goal to another goal if they are not inconsistent each other.

Definition 7 The entailment $\Xi \subseteq G \times G$ is a binary relation on G, defining which goals entail other goals. Formally,

$$\Xi = \{(g_i, g_j) \quad i, j = 1, ..., |G| \; : \; g_i \rightarrow g_j\} . \tag{4}$$

Definition 8 The precondition set $\Pi \subseteq G \times G$ is a binary relation on G, defining which goals are precondition of other goals. Formally,

$$\Pi = \{(g_i, g_j) \quad i, j = 1, ..., |G| \; : \; g_i \mapsto g_j\} . \tag{5}$$

Definition 9 The dependence $\Delta \subseteq G \times G$ is a binary relation on G, defining which goals depend on other goals. Formally,

$$\Delta = \{(g_i, g_j) \quad i, j = 1, ..., |G| \; : \; g_i \hookrightarrow g_j\} . \tag{6}$$

Finally, on the basis of the above properties and relations we can now define the structure of the *goal model* of PRACTIONIST agents as follows

$$GM = \langle G, \; \Gamma, \; \Gamma', \; \Xi, \; \Pi, \; \Delta \rangle \tag{7}$$

where:

- G is the set of goals the agent could pursue;
- Γ is the *inconsistency* relation among goals;
- Γ' is the *preference* relation among inconsistent goals;
- Ξ is the *entailment* relation among goals;
- Π is the *precondition* relation among goals;
- Δ is the *dependence* relation among goals.

IV. Reasoning about goals

In this section we show how the goal elements previously defined are used by PRACTIONIST agents when reasoning about goals during their deliberation process and the means-ends reasoning. We also highlight the actual relations between them and mental attitudes, i.e. desires and intentions.

In PRACTIONIST agents goals and their properties are defined on the basis of what agents believe. Thus, an agent will believe that a goal $g = \langle \sigma_g, \; \pi_g \rangle$ has succeeded if it believes that its success condition σ_g is true. The same holds for the other properties.

It is important to note that, in PRACTIONIST, desires and intentions are mental attitudes towards goals, which are in turn considered as descriptions of objectives. Thus, referring to a goal, an agent can just relate it to a *desire*, which it is not committed to because of several possible reasons (e.g. it believes that the goal is not possible). On the other hand, a goal can be related to an *intention*, that is the agent is actually and actively committed to pursue it.

Let $GM = \langle G, \; \Gamma, \; \Gamma', \; \Xi, \; \Pi, \; \Delta \rangle$ be a *goal model* of a PRACTIONIST agent α and, at a given time, $G' \subseteq G$ be the set of its active goals, which are those goals that the agent is already committed to.

Suppose that α starts its deliberation process and generates the goal $g = \langle \sigma_g, \; \pi_g \rangle$ as an option. Therefore the agent would like to commit to g, that is its *desire* is to bring about the goal g. However, since an agent will not be able to achieve all its desires, it performs the following process in the context of its deliberation phase (figure 2): the agent checks if it believes that the goal g is *possible* and not *inconsistent* (see definition 1) with active goals (belonging to G').

If both conditions hold the desire to pursue g will be promoted to an *intention*. Otherwise, in case of inconsistency among g and some active goals, the desire to pursue g will become an intention only if g is preferred to such inconsistent goals, which will in turn be dropped.

In any case, if the desire to pursue g is promoted to an *intention*, before starting the means-ends reasoning, the agent α checks if it believes that the goal g *succeeds* (that is, if it believes that the success condition σ_g holds) or whether the goal g is entailed (see definition 2) by some of the current active goals. In case of both above conditions do not hold, the agent will perform the means-ends reasoning, by either selecting a plan from a fixed plan library or dynamically generating a plan and finally executing it (details on this means-ends reasoning can be found in [10]).

Indeed, if the goal g succeeds or is entailed by some current active goals (i.e. some other means is working to achieve a goal that entails the goal g), there is no reason to pursue it. Therefore, the agent does not need to make any means-ends reasoning to figure out how to pursue the goal g.

Otherwise, before starting the means-ends reasoning, if some declared goals are precondition for g, the agent will first desire to pursue such goals and then the goal g.

In the PRACTIONIST framework, as a default, an agent will continue to maintain an intention until it believes that either such an intention has been achieved or it is no longer possible to achieve the intention. This commitment strategy to intention is called *single-minded commitment* [13]. In order to perform such a behaviour, the agent continuously checks if it believes that the goal g has just succeeded and that the goal g is still possible.

Moreover the agent checks if some dependee goal does not succeed. If so, it will desire to pursue such a goal and then continue pursuing the goal g. When all dependee goals succeed, the agent resumes the execution of the plan.

In order to be able to recover from *plan failures* and try other means to achieve an intention, if the selected plan fails or is no longer appropriate to achieve the intention, then the agent selects one of applicable *alternative plans* within the

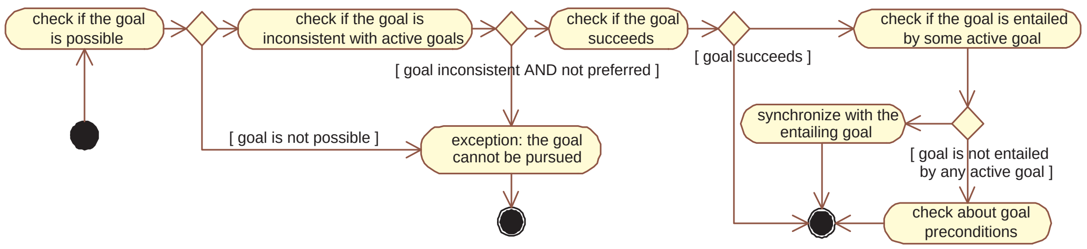

check if the goal is possible
check if the goal is inconsistent with active goals
check if the goal succeeds
check if the goal is entailed by some active goal
[goal inconsistent AND not preferred]
[goal succeeds]
synchronize with the entailing goal
[goal is not possible]
exception: the goal cannot be pursued
[goal is not entailed by any active goal]
check about goal preconditions

Fig. 2. Reasoning about goals: the deliberation phase.

same intended means and executes it.

If none of the alternative plans was able to successfully pursue the goal g, the agent take into consideration the goals that *entail* g. Thus the agent selects one of them and considers it as an option, processing it in the way described in this section, from deliberation to means-ends reasoning.

If there is no plan to pursue alternative goals, the achievement of the intention has failed, as the agent has not other ways to pursue its intention. Thus, according to agents beliefs, the goal was *possible*, but the agent was no able to pursue it (i.e. there are no plans).

V. The Support for the Goal Model in the PRACTIONIST Framework

In order to provide the PRACTIONIST framework with the support for the definition/handling of agent goal models and the capabilities for reasoning about goals, we identified and fulfilled the following requirements:

- registration of the goals that each agent could try to pursue during his life cycle;
- registration of the relations among such goals;
- checking whether two goals are inconsistent and which the preferred one is (if any);
- getting the list of goals that entail a given goal;
- getting the list of goals that are precondition of a given goal;
- getting the list of goals which a given goal depends on.

A proper ad-hoc search algorithm explores the goal model and answers the queries, on the basis of both declared and implicit relations. Indeed, implicit relations (especially inconsistence and entailment) can be inferred from the semantics of some built-in goals, such as state goals (e.g. $achieve(\varphi)$, $cease(\varphi)$, $maintain(\varphi)$, and $avoid(\varphi)$, where φ is a closed formula of FOL). Therefore, the goal reasoner takes into account implicit relations such as $achieve(\varphi)\perp achieve(\neg\varphi)$, $achieve(\varphi)\perp cease(\varphi)$, $maintain(\varphi)\perp avoid(\varphi)$, and so forth.

Figure 3 shows the actual structure of the `GoalModel` that each agent owns (`PRACTIONISTAgent` is the abstract class that has to be extended when developing PRACTIONIST agents). Such a model stores information about declared goals (with their internal properties, i.e. success and possibility condition) and the four types of relations these goals are involved in. Specifically the interface `GoalRelation` provides the super interface for all goal relations supported by the PRACTIONIST framework (i.e. `EntailmentRel`, `InconsistencyRel`, `DependencyRel`, and `PreconditionRel`) and defines the operation `verifyRel`, whose purpose is to check each specific relation.

In order to exploit the features provided by the goal model and understand if a given goal the agent desires to pursue is inconsistent with or implied by some active goals, the agent must have information about such active goals and whether them are related to either desires or intentions. Therefore, each PRACTIONIST agent owns an `ActiveGoalsHandler` component, which, with the aid of the `GoalModel`, has the responsibility of keeping track of all executing intended means stacks with the corresponding waiting and executing goals and managing requests made by the agent.

Thus, at any given time, the `ActiveGoalsHandler` is aware of current desires and intentions of the agent, referring them to active goals.

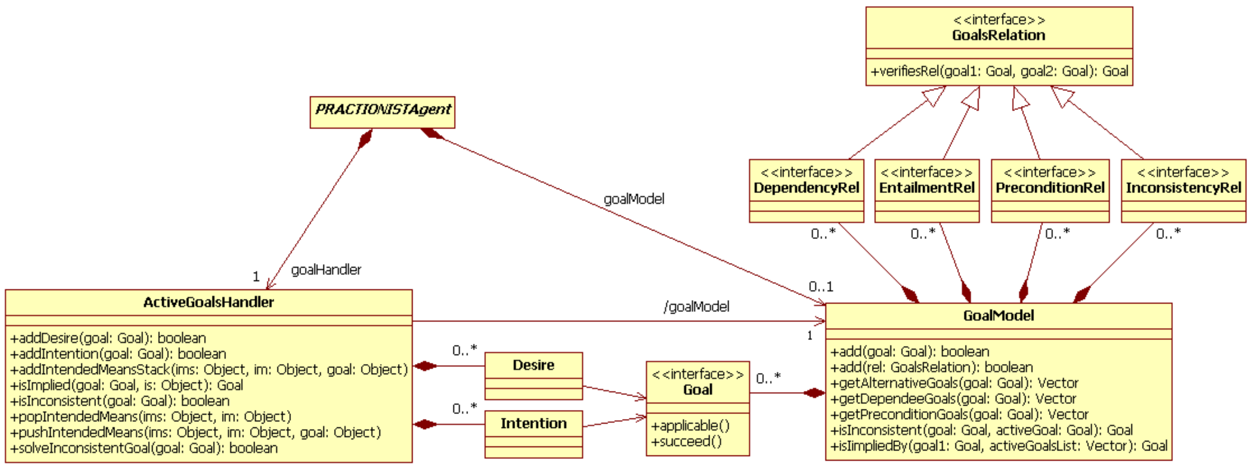
VI. An Example

In this section we present the Tileworld example to illustrate how to use the goal model presented in this paper and the support provided by the PRACTIONIST framework.

The Tileworld example was initially introduced in [14] as a system with a highly parameterized environment that could be used to investigate the reasoning in agents. The original Tileworld consists of a grid of cells on which tiles, obstacles and holes (of different size and point value) can exist. Each agent can move up, down left or right within the grid to pick up and move tiles in order to fill the holes. Each hole has an associated score, which is awarded to the agent that has filled the hole. The main goal of the agent is to score as many points as possible.

Tileworld simulations are dynamic and the environment changes continually over time. Since this environment is highly parameterized, the experimenter can alter various aspects of it through a set of available ”knobs”, such as the rate at which new holes appear (*dynamism*), the rate at which obstacles appear (*hostility*), difference in hole scores (*variability of utility*), and so forth.

Such applications, with a potentially high degree of dynamism, can benefit from the adoption of a goal-oriented design approach, where the abstraction of goal is used to declaratively represent agents' objectives and states of affairs that can be dynamically achieved through some means.



<<interface>>
GoalsRelation
+verifiesRel(goal1: Goal, goal2: Goal): Goal
PRACTIONISTAgent
goalModel
<<interface>>
DependencyRel
<<interface>>
EntailmentRel
<<interface>>
PreconditionRel
<<interface>>
InconsistencyRel
0..*
0..*
0..*
0..*
1
goalHandler
0..1
ActiveGoalsHandler
+addDesire(goal: Goal): boolean
+addIntention(goal: Goal): boolean
+addIntendedMeansStack(ims: Object, im: Object, goal: Object)
+isImplied(goal: Goal, is: Object): Goal
+isInconsistent(goal: Goal): boolean
+popIntendedMeans(ims: Object, im: Object)
+pushIntendedMeans(ims: Object, im: Object, goal: Object)
+solveInconsistentGoal(goal: Goal): boolean
/goalModel
1
GoalModel
+add(goal: Goal): boolean
+add(rel: GoalsRelation): boolean
+getAlternativeGoals(goal: Goal): Vector
+getDependeeGoals(goal: Goal): Vector
+getPreconditionGoals(goal: Goal): Vector
+isInconsistent(goal: Goal, activeGoal: Goal): Goal
+isImpliedBy(goal1: Goal, activeGoalsList: Vector): Goal
0..*
Desire
0..*
Intention
<<interface>>
Goal
+applicable()
+succeed()
0..*

Fig. 3. The structure of the support for the goal model in the PRACTIONIST framework.

Figure 4 shows the Tileworld environment, where new agents can be added or removed and the corresponding parameters can be dynamically changed.

In our Tileworld demonstrator two types of agents were developed, the Tileworld Management Agent (TWMA) and the Tileworld Player Agent (TWPA): the former is the agent that manages and controls the environment, by creating and destroying tiles, holes and obstacles, according to the parameters set by the user; the latter is the agent moving within the grid and whose primary goal is to maximize its score by filling holes with tiles. A player agent does not get any notification about the environment changes (i.e. by the management agent), but it can ask such an information (e.g. what the current state of a cell is) by means of sensing actions, in order to adopt the best strategy on the basis of the current state of the environment. In fact, for each state of the environment (e.g., static, dynamic, very dynamic, etc.) at least a strategy is provided. All the strategies are implemented through plans that share the same goal and differ for their operative conditions (i.e. the context).

It should be noted that, since PRACTIONIST agents are endowed with the ability of dynamically building plans starting from a given goal and a set of available actions, some strategies could be generated on-the-fly by taking into account emerging situations.

The player agent has beliefs about the objects that are placed into the grid, its position, its score, the state of the environment, etc.

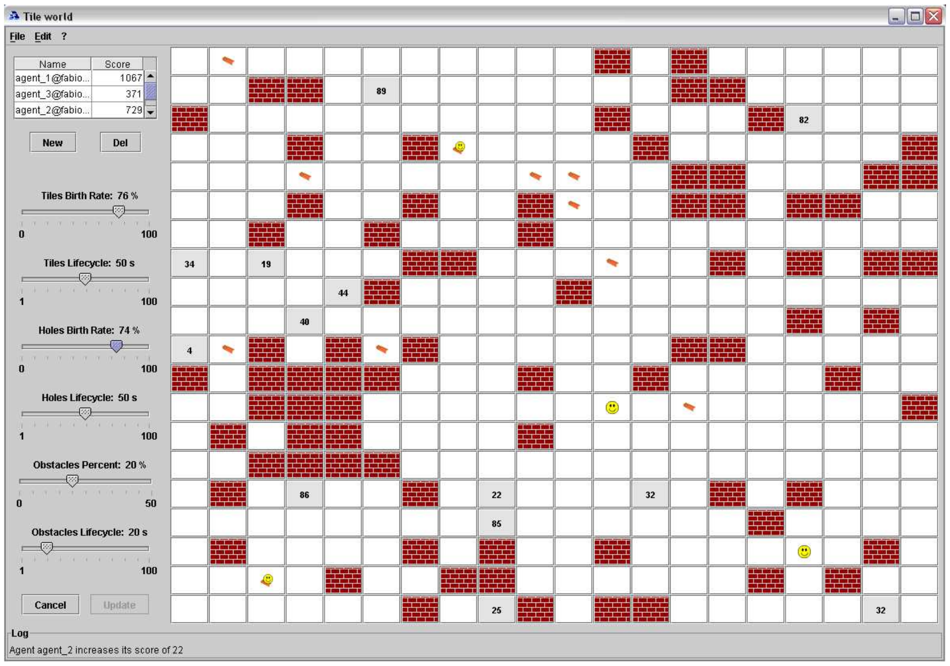
The TWPA top level goal is to score as many points as possible, but to do this, it has to register itself with the manager, look for the holes and for the tiles, hold a tile, and fill a hole.

We designed the TWPA by adopting the goal-oriented approach described in this paper and directly implemented its goal-related entities (i.e. goals and relations) thank to the support provided by the PRACTIONIST framework. In figure 5 a fragment of the goal model of the TWPA is shown as a UML class diagram with dependencies stereotyped with the name of the goal relations. Actually some relations only hold under certain condition and the diagram does not show such details.

According to the diagram, the TWPA has to be registered with the TWMA before increasing its score (the goal `ScorePoints` *depends on* the goal `RegisterWithManager`). Moreover, in order to score points, the TWPA has to fill as many holes as possible (the goal `FillHole` *entails* the goal `ScorePoints`). But, in order to fill a hole, the TWPA has to hold a tile and find a hole (the goal `FillHole` *depends on* the goal `HoldTile` and requires the goal `FillHole` as *precondition*); finally, the TWPA has to find the tile to hold it (the goal `HoldTile` has the goal `FindTile` as a precondition).

According to the above-mentioned description, the following source code from the TWPAgent class shows how goals and relations among them are added to the agent and thus how to create the goal model through the PRACTIONIST framework:

```
protected void initialize()
{
 ...
 GoalModel gm = getGoalModel();

 // Goal declaration
 gm.add(new RegisterWithManager());
 gm.add(new ScorePoints());
 gm.add(new HoldTile());
 gm.add(new FindTile());
 gm.add(new FillHole(getBeliefBase()));
 gm.add(new FindHole());

 // relations among goals
 gm.add(new Dep_ScorePoints_RegisterWithManager());
 gm.add(new Ent_ScorePoints_FillHole());
 gm.add(new Dep_FillHole_HoldTile());
 gm.add(new Pre_HoldTile_FindTile());
 gm.add(new Pre_FillHole_FindHole());

 ...
}
```



Name	Score
agent_1@fabio...	1067
agent_3@fabio...	371
agent_2@fabio...	729

Tiles Birth Rate: 76 %
Tiles Lifecycle: 50 s
Holes Birth Rate: 74 %
Holes Lifecycle: 50 s
Obstacles Percent: 20 %
Obstacles Lifecycle: 20 s

```
Agent agent_2 increases its score of 22
```

Fig. 4. The Tileworld environment.

In order to better understand how the above-mentioned relations are implemented, the following source code shows the precondition relation among the goals `HoldTile` and `FindTile`:

```
public class Pre_HoldTile_FindTile
      implements PreconditionRel
{
  public Goal verifyRel(Goal goal1, Goal goal2)
   {
    if((goal1 instanceof HoldTile) &&
        (goal2 instanceof FindTile))
           return new FindTile;

     return null;
  }
  ...
}
```

When the player agent desires to pursue a goal, it checks if this goal is involved in some relations and in that case it reasons about them during the deliberation, means-ends, and intention reconsideration processes. Thus, developers only need to specify goals and relations among them at the design time.

As an example, when the TWPA desires to fill a hole (i.e. `FillHole`), according to the defined goal model and the semantics described in section 2, the agent automatically will check if it just holds a tile (i.e. `HoldTile`); if not, such a goal will be desired. On the other hand, the agent will check if it has found a hole (i.e. `FindHole`) and again, if not, it will desire that.

Moreover, when pursuing the goal `FillHole`, the agent will continuously check the success of all its dependee goals (i.e. `HoldTile`) and *maintain* them in case of failure.

It should be noted that the plan to pursue the goal `FillHole` does not need to include the statements to desire either the dependee (i.e. `HoldTile`) or precondition (i.e. `FindHole`) goals, as shown in the following code fragment.

```
public class FillHolePlan extends GoalPlan
{
  public void body() throws PlanExecutionException
  {
    String posPred = "pos(obj1: X,obj2: Y)";
    AbsPredicate pos =
      getBeliefBase().retrieveAbsPredicate(
        AbsPredicateFactory.create(posPred));

    int xPos = pos.getInteger("obj1");
    int yPos = pos.getInteger("obj2");

    doAction(new ReleaseTileAction(xPos, yPos,
       twaServer.getHoleValue(xPos, yPos)));
    ...
    }
  ...
}
```

The Tileworld domain highlights how the PRACTIONIST goal model is particularly adequate to model dynamic environments in a very declarative manner.

VII. CONCLUSIONS AND FUTURE WORK

In the PRACTIONIST framework, desires and intentions are mental attitudes towards goals, which are in turn considered as descriptions of objectives.

In this paper we described how a declarative representation of goals can support the definition of desires and intentions in PRACTIONIST agents. It also supports the detection and the resolution of conflicts among agents' objectives and activities. This results in a reduction of the gap between BDI theories and several available implementations.

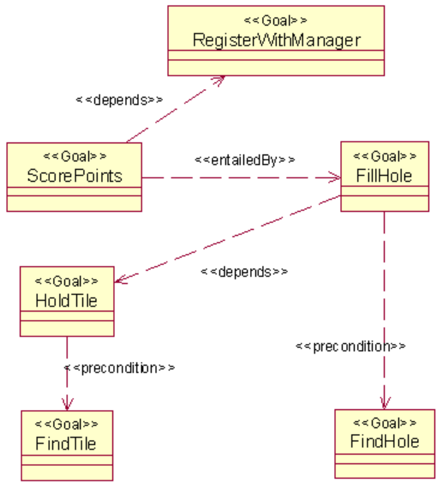


Fig. 5. TWPA's goal model.

We also described how goals and relations are used by PRACTIONIST agents during their deliberation process and the execution of their activities; particularly it is described how agents manages these activities by using the support for the goal model shown in the previous sections.

It should be noted that, unlike several BDI and non-BDI agent platforms, the PRACTIONIST framework supports the declarative definition of goals and the relations among them, as described in this paper. This provides the ability to *believe* if goals are impossible, already achieved, incompatible with other goals, and so forth. This in turn supports the *commitment strategies* of agents and their ability to autonomously drop, reconsider, replace or pursue intentions related to active goals.

The ability of PRACTIONIST agents to reason about goals and the relations among them (as described in section IV) lets programmers implicitly specify several behaviours for several circumstances, without having to explicitly code such behaviours, letting agents figure out the right activity to perform on the basis of the current state and the relations among its potential objectives.

Goals can be adopted throughout the whole development process. Thus, we are defining a development methodology where goals play a central role and maintain the same semantics from early requirements to the implementation phase.

As a part of our future strategy, we aims at extending the proposed model with further properties of goals and relations among them. Finally, we aim at applying the concepts and the model described in this paper in the development of real-world applications based on BDI agents.

Acknowledgments. This work is partially supported by the Italian Ministry of Education, University and Research (MIUR) through the project PASAF.

REFERENCES

[1] A. Lapouchnian, S. Liaskos, J. Mylopolous, and Y. Yu, "Towards requirements-driven autonomic systems design," *Proceedings of the 2005 workshop on Design and evolution of autonomic application software*, pp. 1–7, 2005, aCM Press, New York, NY, USA.

[2] A. S. Rao and M. P. Georgeff, "BDI agents: from theory to practice," in *Proceedings of the First International Conference on Multi—Agent Systems*. San Francisco, CA: MIT Press, 1995, pp. 312–319. [Online]. Available: http://www.uni-koblenz.de/~fruit/LITERATURE/rg95.ps.gz

[3] M. E. Bratman, *Intention, Plans, and Practical Reason*. Cambridge, MA: Harvard University Press, 1987.

[4] M. Winikoff, L. Padgham, J. Harland, and J. Thangarajah, "Declarative & procedural goals in intelligent agent systems," in *KR*, 2002, pp. 470–481.

[5] P. Busetta, R. Rnnquist, A. Hodgson, and A. Lucas, "Jack intelligent agents - components for intelligent agents in java," 1999.

[6] M. J. Huber, "Jam: A bdi-theoretic mobile agent architecture." in *Agents*, 1999, pp. 236–243.

[7] A. S. Rao, "AgentSpeak(L): BDI agents speak out in a logical computable language," in *Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, R. van Hoe, Ed., Eindhoven, The Netherlands, 1996. [Online]. Available: citeseer.ist.psu.edu/article/rao96agentspeakl.html

[8] K. V. Hindriks, F. S. D. Boer, H. W. van der, and J. J. Meyer, "Agent programming in 3APL," *Autonomous Agents and Multi-Agent Systems*, vol. 2, no. 4, pp. 357–401, 1999, publisher: Kluwer Academic Publishers, Netherlands.

[9] L. Braubach, A. Pokahr, W. Lamersdorf, and D. Moldt, "Goal representation for bdi agent systems," in *Second International Workshop on Programming Multiagent Systems: Languages and Tools*, 7 2004, pp. 9–20.

[10] V. Morreale, S. Bonura, G. Francaviglia, M. Cossentino, and S. Gaglio, "Practionist: a new framework for bdi agents," in *Proceedings of the Third European Workshop on Multi-Agent Systems (EUMAS'05)*, 2005, p. 236.

[11] F. Bellifemine, A. Poggi, and G. Rimassa, "JADE - a FIPA-compliant agent framework," in *Proceedings of the Practical Applications of Intelligent Agents*, 1999. [Online]. Available: http://jmvidal.cse.sc.edu/library/jade.pdf

[12] B. F. Chellas, *Modal Logic: An Introduction*. Cambridge: Cambridge University Press, 1980.

[13] A. S. Rao and M. P. Georgeff, "Modeling rational agents within a BDI-architecture," in *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning*. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA, 1991, pp. 473–484. [Online]. Available: http://citeseer.nj.nec.com/rao91modeling.html

[14] M. E. Pollack and M. Ringuette, "Introducing the tileworld: Experimentally evaluating agent architectures," *National Conference on Artificial Intelligence*, pp. 183 – 189, 1990.