DEVELOPING APPLICATIONS WITH HDS

Enrico Franchi, Agostino Poggi, Michele Tomaiuolo
Dipartimento di Ingegneria dell'Informazione
Università degli Studi di Parma
Parma, Italy

Abstract—**HDS (Heterogeneous Distributed System) is a software framework that tries to simplify the realization of distributed applications and, in particular, of multi-agent systems, by merging the client-server and the peer-to-peer paradigms and by implementing all the interactions among the processes of a system through the exchange of a kind of message that allows the implementation of a large set of communication protocols and languages. HDS has been experimented in the realization of systems for information retrieval and for the analysis of social networks.**

Keywords - software framework; layered framework; inmformation retrieval; social networks.

I. Introduction

In the early 1990s, Multi-Agent Systems (MAS) were put forward as a promising paradigm both for developing complex distributed systems and for supporting the interoperability among legacy systems [1][2]. Over the years, MAS researchers have developed a wide body of models, techniques and methodologies for developing complex distributed systems, have realized several effective software development tools, and have contributed to the realization of several successful applications.

However, even if today's software systems are more and more characterized by a distributed and multi-actor nature, that lends itself to be modeled and realized taking advantage of MAS techniques and technologies, very few space in software development is given to the use of such techniques and technologies. It is due to several reasons [3][4][5][6][7]. One of the most important reasons is that software developers usually have limited knowledge about MAS technologies and solutions: it is mainly because of the lack of references to the results of MAS research outside the MAS community. Moreover, even when there is a good knowledge about MAS, software developers do not evaluate the possibility of their use because: i) they believe that multi-agent approaches are not technically superior to traditional approaches (i.e., there are not problems where a MAS approach cannot be replaced by a non-agent approach), and ii) they consider MAS approaches too sophisticated and hard to understand and to be used outside the research community.

Therefore, it is possible to state that the MAS community has yet to demonstrate the significant benefits of using agent-oriented approaches to solve complex problems, but also that some efforts should be done for facilitating the use and the integration of MAS technologies and solutions in mainstream software development technologies and solutions. In particular, MAS developers should avoid themselves considering MAS solutions to be a "panacea" for all the kinds of system. Therefore, a MAS should be realized only when the components of a system must express the typical features (i.e., proactiveness, sociality and goal orientation) that distinguish a software agent from another software component.

In this paper, we present a software framework, called HDS, whose goal is to simplify the realization of distributed applications taking advantage of multi-agent model and techniques and to provide an easy way for the integration between MAS and non-agent based systems. The next section describes the main features of the HDS software framework. Section three and four discuss about the experimentation of such a framework for the development of information retrieval and social network applications. Finally section five concludes the paper sketching some future research directions.

II. HDS

HDS (Heterogeneous Distributed System), is a software framework that has the goal of simplifying the realization of distributed applications by merging the client-server and the peer-to-peer paradigms and by implementing all the interactions among the processes of a system through the exchange of typed messages. In particular, HDS provides both proactive and reactive processes (respectively called actors and servers) and an application can be distributed on a (heterogeneous) network of computational nodes (from now on called runtime nodes).

The HDS software framework model is based on two layers called, respectively: concurrency and runtime layers. While the first layer defines the elements that an application developer must directly use for realizing applications, the second layers, besides providing the services that enable the creation of the elements of the concurrency layer and their interaction, abstracts the use of different technologies for realizing distributed applications on networks of heterogeneous devices connected through a set of different communication transport protocols.

A. *Concurrency Layer*

The concurrency layer is based on six main elements: process, description, selector, message, content and filter.

A *process* is a computational unit able to perform one or more tasks taking, if necessary, advantage of the tasks provided by other processes. To facilitate the cooperation among processes, a process can advertise itself making available to the other processes its description. Usually a *description* contains the process identifier, the process type and the data that have been used for its initialization; however, a process may introduce some additional information in its description.

A process can be either an actor or a server. An *actor* is a process that can have a proactive behavior and so can start the execution of some tasks without the request of other processes. A *server* is a reactive process that is only able to perform tasks in response of the request of other processes.

A process can interact with the other processes through the exchange of messages based on one of the following three types of communication: i) synchronous communication, the process sends a message to another process and waits for its answer; ii) asynchronous communication, the process sends a message to another process, performs some actions and then waits for its answer and iii) one-way communication, the process sends a message to another process, but it does not wait for an answer. In particular, while an actor can start all the three previous types of communication with all the other processes, a server can either respond to the requests of the other processes or can delegate the execution of such requests to some other processes.

Taking advantage of the registry service provided by the runtime layer, a process has also the ability of discovering the other processes of the application. In particular, a process can: i) check if an identifier is bound to a process of the application, ii) get the identifiers of the other processes of its runtime node, and iii) get the identifiers of the processes of the application whose description satisfies some constraints. The last capability is possible, because, a process can create a special type of object, called *selector*, that define some constraints on the information maintained by the process descriptions (e.g., the process must be of a specific type, the process identifier must have a specific prefix or suffix), then the process sends such a selector to the registry service provided by the runtime layer, and the registry service applies the constraints defined by the selector on the information of the registered process descriptions and sends to the processes the identifiers of the processes that satisfy the constraints defined by the selector.

As we wrote above, processes interact with each other through the exchange of messages. A *message* contains the typical information used for exchanging data on the net, i.e., some fields representing the header information, and a special object, called content, that contains the data to be exchanged. In particular, the *content* object is used for defining the semantics of messages (e.g., if the content is an instance of the Ping class, then the message represents a ping request and if the content is an instance of the Result class, then the message contains the result of a previous request). In particular, the message model defined by the concurrency layer allows the implementation of a large set of communication protocols and languages. In fact, the traditional client-server protocol can be realizing associating the request and response data to the message content element and the most known agent communion language, i.e., KQML and FIPA ACL [8], can be realized by using the content element for the representation of ACL messages.

Normally, a process can interact with all the other processes of the application and the sending of messages does not involve any operation that is not related to the delivery of messages to the destination; however, the presence of message filters can modify the normal delivery of messages. A message filter is a composition filter [9] whose primary scope is to define the constraints on the reception/sending of messages; however, it can also be used for manipulating messages (e.g., their encryption and decryption) and for the implementation of replication and logging services. Figure 1 shows the flow of messages towards inside a process.

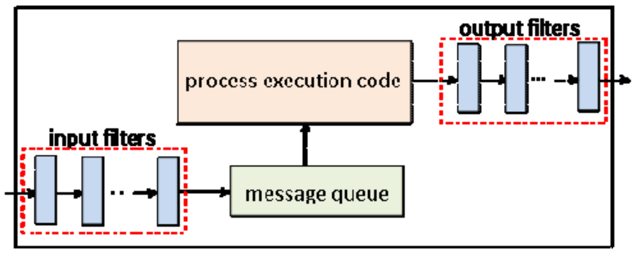

Figure 1. Flow of the messages inside a process.

The runtime layer associates two lists of message filters with each process: the ones of the first list, called input message filters, are applied to the input messages and the others, called output message filters, are applied to the output messages. When a new message arrives or must be sent, the relative message filters are applied to it in sequence until a message filter fails; therefore, such a message is stored in the input queue or is sent only if all the message filters have success.

B. *Runtime Layer*

The main goal of the runtime layer is to allow the use of different technologies for realizing distributed applications on networks of heterogeneous devices connected through a set of different communication transport protocols, but abstracting such technologies through a tiny API towards the concurrency layer. In particular, the runtime layer defines a set of services that can be used by the concurrency layer and a set of interfaces that must be implemented for integrating a new technology and using it for providing the services provided by the runtime layer.

The main element of the runtime layer is the reference. A *reference* is a proxy of the process that makes transparent the communication respect to the location of the process and the technologies connecting the reference with its process. The duty of a reference is to allow the insertion of messages in the queue of its process. Therefore, when a process wants to send a message to another process, it must obtain the reference to such a process and then use it for putting the message in the input queue of the other process.

The access to the reference of a process is possible through the use of the registry service. This service is provided by the runtime layer to the processes of an application and allows: the binding and unbinding of the processes with their identifiers, description, and references, the retrieval of a reference on the basis of the process identifier and the listing of sets of identifiers of the processes of an application by using, if necessary, some selectors.

The runtime layer has also the duty of creating processes and their related references. In fact, it provides a factory service that allows a process of an application to create other processes by proving to the runtime layer the qualified name of the class implementing the process and its initialization list.

Finally, the runtime layer provides a filtering service that allows the management of the list of message filters associated with the processes of an application. In fact a process cannot modify any list of message filters. Therefore, taking advantage of the filterer service, a process can modify the lists of its message filters, but can also drive the behavior of some other processes by modifying their message filter lists.

C. *Implementation*

The HDS software framework has been realized taking advantage of the Java programming language. While the runtime layer has been implemented for providing the remote delivery of messages through both Java RMI and JMS [11] communication technologies, the concurrency layer provides: i) a message implementation, ii) four abstract classes that implement the application independent parts of actors, servers, selectors and filters, and iii) a set of abstract and concrete content classes useful for realizing the typical communication protocols used in distributed applications. In particular, HDS provides a client-server implementation of all the protocols used by processes for accessing to the services of the runtime layer and a complete implementation of the FIPA ACL coupled with an abstract implementation of the "roles" involved in the FIPA interaction protocols. Moreover, for simplified the deployment of application, current HDS implementation provides a software tool that allows the deployment of applications through the use of a set of configuration files.

Moreover, the current implementation provides a task-based library (similar to the behavior library provided by the JADE software framework [12][13][14]) that allows the definition of the “behavior” of a HDS process as composition of a set of predefined tasks. In particular, the main feature of such a task based library is that it allows the realization of both applications, where processes have their own execution thread and perform concurrently their tasks, and application, where groups of processes share the same thread and a scheduler executes sequentially the tasks of such processes. This feature is very important because it allows the realization of applications supporting the concurrent execution of thousands of processes in each runtime node and because it makes HDS suitable to be used as a simulation tool.

III. USING HDS FOR INFORMATION RETRIEVAL

Building over our previous experiences on similar projects on JADE, one of the first applications we have built on HDS is a system that supports the sharing of information among a community of users connected through the Internet. Even if nowadays the Web is a powerful tool for getting information about any kind of topic, it assigns a passive role to a large part of its users. In respect to Web search engines, this system enhances the search through domain ontologies, avoids the burden of publishing the information on a web server and guaranties a controlled and dynamic access to the information. Those advantages are possible thanks to the use of peer-to-peer technologies coupled with a multi-agent approach, to allow the active sharing of information over the Internet, among the members of a community.

Multi-agent systems have always been considered one of the most important ingredients for the development of distributed information management systems and for proving the different services needed in such systems. In fact, HDS allowed us to develop a multi-agent system, by composing various runtime platforms, connected through the Internet and leveraging an overlay network. Each runtime platform acts as a “peer” of the system and is based on four components, realized as processes of various types: a personal assistant, a repository manager, an information finder, an information pusher, and a directory facilitator.

A personal assistant (PA) is an HDS actor process that allows the interaction between a user and the rest of the system. This agent receives the user’s queries, forwards them to the available information finders and presents the results to the user. Moreover, as a proactive behavior, a PA allows the user to be informed about the new information that other users made available and that may be of her/his interest. Finally, a PA maintains the information that a user may share allowing her/him to add and remove information in a repository where information is partitioned on the basis of the topics of interest of the user.

A repository manager (RM) is a server process that builds and maintains both the indexes for searching information and the ontologies describing the topics of interest of its user. Each time the user adds or removes some information, the RM updates the corresponding index and ontology.

An information finder (IF) is a server process that searches information on the repository contained into the computer where it lives and provides this information both to its user and to other users of the system. An IF receives users’ queries, finds appropriate results, on the basis of both the queries and the topic ontology, and filters them on the basis of its user’s policies (e.g., the results from non-public folders are not sent to other users).

An information pusher (IP) is an HDS actor process that monitors the changes in the local repository and pushes the new information to the PA of the users whose previous queries match such information.

Finally, the directory facilitator (DF) is responsible to register the agent platform in the network. The DF, implemented as a server process, is also responsible to inform the processes of its platform about the existence and location of the processes that live in the other runtime platforms available on the network (e.g., a PA can ask about the address of the active IF processes).

The exchange of information among the users of the system is driven by the creation of a search index and of an ontology for each topic. The search index allows ranking information on the basis of the terms contained in a query. The ontology allows identifying additional information on the basis of the terms contained in the ontology that have some semantic relationships (i.e., synonyms, hyponyms, hypernyms, meronyms and holonyms) with the terms contained in the query. Both the search index and the ontology are automatically built by the RM on the basis of the information stored in the topic repository.

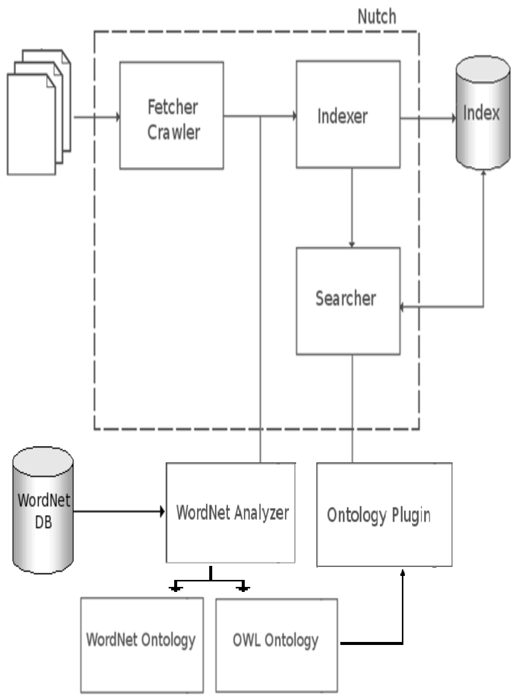

Figure 2. Indexing and ontology management subsystem.

Even if there are some specific tools and software libraries for searching information in a local repository (see, for example, Beagle [15] and Google Desktop Search [16]), we adapted Nutch [17], an open source web-search software, for searching the local repository. It has been done because it is very easy to develop Nutch plugins for extending its capabilities (we used this feature for using its term extraction module for building the topic ontologies) and because a Nutch plugin, that extends keywords based search through the use of OWL ontologies, is available [18].

As introduced above, topic ontologies are built by a Nutch plugin. This plugin receives the terms extracted from the information to be indexed by the Nutch software. Then, accessing the WordNet lexical database [19][20] though the use of the JAWS Java software library [21], for each term it identifies the top terms of the ontology and the other terms extracted from the information that have some semantic relationships (i.e., synonyms, hyponyms, hypernyms, meronyms and holonyms). At the end of this process, all the terms that have a semantic distance greater than the one fixed by the user are removed and then the WordNet ontology is saved as an OWL file.

Figure 2 shows a graphical description of the work done by the Nutch core software and by its two plugins for indexing, building the topic ontologies and using them for searching information.

The information stored into the different repositories of the network is not accessible to all the users of the system in the same way. In fact, it's important to avoid the access to private documents and personal files, but also to files reserved to a restricted group of users (e.g.: the participants of a project). The system takes care of users' privacy allowing the access to the information on the basis of the identity, the roles and the attributes of the querying user, as defined into a local knowledge base of trusted users. In this case, it is the user that defines who and in which way can access her/his information. Moreover, the user can also grant the access to unknown users by enabling a certificate based delegation, built on a network of the users registered into the community. In this sense, the system completely adheres to the principles of trust management.

The definition of roles and attributes is made in a local namespace, and the whole system is, in this regard, completely distributed. Links among different local namespace can be explicitly defined by issuing appropriate certificates. In this sense, local names are the distributed counterpart of roles in role based access control frameworks.

Usual agent environments, as well as most distributed systems, do not provide any support for managing advertisements in a completely distributed fashion, neither they implement some overlay structure. For this reason, in previous works we integrated JXTA technologies with multi-agent systems, in a way to adhere to relevant FIPA specifications, in particular those regarding the Agent Discovery Service and the JXTA Discovery Middleware. At present, we're integrating DHT mechanisms directly into HDS, for leveraging more widespread technologies as Kademlia.

IV. USING HDS FOR SOCIAL NETWORKS

A social network is a social structure constituted by individuals and by their mutual connections. A social network can be represented as a graph where the individuals are the vertices and the connections are the edges. A Social Network System (SNS) is a site allowing users to: i) construct a profile within a bounded system; ii) articulate a list of other users with whom they share a connection; iii) view and traverse their list of connections and those made by users within the system [22]. We also expect a SNS to suggest proactively possible acquaintanceships among users, using the information in user profiles (or other user provided data) according to user specified policies.

Today's ever-increasing diffusion of online social networking sites boosted researchers' interest in social networks in general and specifically in social network analysis. The inherently large scale of such services calls for automated techniques capable of promoting their potentials to greater levels in terms of offered functionality and performance. Such automated techniques are still far from real-world practice because the impact of a novel algorithm (e.g., a friendship-discovery algorithm [23]) cannot be easily assessed. This is the reason why there is the necessity for effective tools to study, experiment and validate innovative techniques that are capable of providing concrete evaluation on the net results on the introduction of novel proposals into a social networking system.

Specifically, we need realistic synthetic social networks to bootstrap the algorithms and protocols we are developing. Ideally, we would like to use many different social networks to test our work, and consequently we need software tools able to generate the required amount of networks. We searched for such models in the literature and although researchers created many models to generate complex networks, only few are able to generate meaningful synthetic social networks [24]. These models are mostly probabilistic models somewhat rooted in the Erdös-Renyi tradition [25] we believe, instead, that novel agent-based models could be easier to develop, to study and could more easily reproduce the complex social processes that led to the formation of a social network from the low-level individual interactions.

Although the models we reviewed in [24] are not agent-based, they can be easily translated in an agent-based framework as discrete event simulations. Further details on the rendition of such probabilistic models as agent-based models are given in [24].

In order to easily test agent-based models, we created an extensible agent-based social network generation system. All communications between the agents of the systems are made through message passing. The agents have their own thread of control and explicitly receive and send messages when they consider it appropriate.

In the present version, there is a clock, which sends *Tick* messages to interested agents, so that ages in the simulations are clearly divided. When the simulation ends, the clock sends an *EOW* message informing the agents that they should start the cleanup process.

A specific agent performs the selection of the nodes to be activated and we refer to that agent as the activator. In fact, the selection of the nodes to activate is only one of its tasks, since it is also responsible to select the agents to be destroyed and to feed those to be created with the appropriate initialization parameters. The specification of the activator's tasks is done providing: i) a node selector object to select the nodes to activate; ii) a node selector object to select the nodes to destroy; and iii) a node factory object to determine the class and the parameters of the nodes to be created.

When the activator has selected the nodes to be activated, it sends them an *Activate* message. When a node agent receives an *Activate* message, it decides the course of the following actions. When the actions, whichever they are, are finished, the node agent acknowledges the activator that it has completed its task. This way the activator knows when all the actions of a given simulation step are terminated.

The agent-based network generation system is created on the top of HDS. We used composition filters to add a spy agent, which receives all the messages that are relevant to keep an updated view of the network (link or node formation or removal). Consequently implementing the node agents' behavior and the node activator behavior is completely separated from writing and plugging-in code to perform runtime network analysis or to perform any other action on the network itself (e.g., visualize it or save it on file).

It is interesting to outline that the models we have currently implemented in our generation system do not use full agent powers. The reason is that using pre-existing models allowed us to concentrate more on the engineering issues of building such a system. However, it is easy to extend our system with models that make use of pro-active learning node agents. Examples of such improvements could be replacing the probabilistic choice of the node at the other end of a link with a protocol taking into account, for example: i) the number of friends in common, ii) mutual interests, iii) different kind of links other than friendship, iv) gossip.
A further motivating advantage of using HDS instead of existing discrete event simulation engines such as RePast [26], Swarm [27], NetLogo [28] and Mason [29] is that it is easier to support continuous simulations. Most of the existing infrastructure needs not to be changed, *Activate* messages would disappear and node agents would simply pro-actively search for new friendships according to their own schedule.

V. Conclusions

This paper presented HDS, a software framework that has the goal of simplifying the realization of distributed applications by merging the client-server and the peer-to-peer paradigms and by implementing the interactions among all the processes of a system through the exchange of typed messages.

HDS is implemented by using the Java language and its use simplify the realization of systems in heterogeneous environments where computers, mobile and sensor devices must cooperate for the execution of tasks. Moreover, the possibility of using different communication protocols for the exchange of messages between the processes of different computational nodes of an application opens the way for a multi-language implementation of the HDS framework allowing the integration of hardware and software platforms that do not provide a Java support.

HDS can be considered a software framework for the development of any kind of distributed system. Some of its functionalities derive from the one offered by JADE [12][13][14], a software framework that can be considered one of the most known and used software framework for the developing of multi-agent systems. This derivation does not depend only on the fact that some of the people involved in the development of the HDS software framework were involved in the development of JADE too, but because HDS tries to propose a new view of multi-agent systems where the respect of the FIPA specifications are not considered mandatory and ACL messages can be expressed in a way that is more usable by software developers outside the multi-agent system community. This work may be important not only for enriching other theories and technologies with some aspects of multi-agent system theories and technologies, but also for providing new opportunities for the diffusion of both the knowledge and use of multi-agent system theory and technologies.

Current and future research activities are dedicated, besides to continue the experimentation and validation of the HDS software framework in the realization of collaborative services for social network, to the improvement of the HDS software framework. In particular, current activities are dedicated: i) to extend the use of HDS for pervasive applications through the implementation of more sophisticated adaptation services based on message filters taking advantages of the solutions presented by PICO [30] and by PCOM [31] and to improve the interoperability with other systems by automatically mapping the Java classes defining the content of the typed messages into OWL ontologies and by supporting the interaction among system encoding messages into a RDF format.

REFERENCES

[1] M. R. Genesereth and S. P. Ketchpel. Software Agenta, Communications of ACM, 37(7):48-63, 1994.

[2] J. M. Bradshaw. An introduction to software agents. in, Jeffrey M. Bradshaw, Ed, Software Agents, pp. 3-46, MIT Press, Cambridge, MA, 1997.

[3] V. Maříka and J. Lažanský. Industrial applications of agent technologies. Control Engineering Practice, 15(11):1364-1380, 2007.

[4] L. Braubach, A. Pokahr and W. Lamersdorf. A Universal Criteria Catalog for Evaluation of Heterogeneous Agent Development Artifacts. In Proc. of the Sixth Int.Workshop "From Agent Theory to Agent Im-plementation" (AT2AI-6), pp. 19-28, Estoril, Portugal, 2008.

[5] J. McKean, H. Shortery, M. Luckz, P. McBurneyx and S. Willmott. Technology diffusion: analysing the diffusion of agent technologies, Autonomous Agents and Multi-Agent Systems, 17(2):372-396, 2008.

[6] S. A. DeLoach. Moving multi-agent systems from research to practice. Agent-Oriented Software Engineering, 3(4):378-382, 2009.

[7] D. Weyns, A. Helleboogh and T. Holvoet. How to get multi-agent systems accepted in industry? Agent-Oriented Software Engineering, 3(4):383-390, 2009.

[8] Y. Labrou, T. Finin, and Y. Peng. Agent Communication Languages: The Current Landscape. IEEE Intelligent Systems, 14(2):45-52. 1999.

[9] L. Bergmans and M. Aksit. Composing crosscutting concerns using composition filters. Communications of ACM, 44(10):51-57, 2001.

[10] E. Pitt and K. McNiff. Java.rmi: the Remote Method Invocation Guide. Addison-Wesley, 2001.

[11] R. Monson-Haefel and D. Chappell. Java Message Service. O'Reilly & Associates, 2000.

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

[13] F. Bellifemine, G. Caire, A. Poggi and G. Rimassa. JADE: a Software Framework for Developing Multi-Agent Applications. Lessons Learned. Information and Software Technology Journal, 50:10-21, 2008.

[14] JADE. Available from http://jade.tilab.com.

[15] Beagle Team, “Beagle,” 2001, available from http://beagle-project.org.

[16] Google, “About Google Desktop Search,” 2011, available from http://desktop.google.com.

[17] Apache Foundation, “Nutch,” 2011, available from http://nutch.apache.org.

[18] W3C Consortium, “OWL 2 Web Ontology Language Overview,” 2009, available from http:// http://www.w3.org/TR/owl2-overview.

[19] G.A. Miller. WordNet: A Lexical Database for English. Communications of the ACM Vol. 38, No. 11, 1995, pp. 39-41.

[20] Pricetom Universty, “Wordnet,” 2011, available from http://wordnet.princeton.edu.

[21] Southern Methodist University, ”JAWS,” 2011, available from http://lyle.smu.edu/~tspell/jaws.

[22] D. M. Boyd and N. B. Ellison. Social network sites: Definition, history, and scholarship. Journal of Computer-Mediated Communication, 13(1):210–230, 2008.

[23] E. Franchi. A Multi-Agent Implementation of Social Networks. In Proc. of the Undicesimo Workshop Nazionale “Dagli Oggetti agli Agenti” (WOA 2010), Rimini, 2010.

[24] F. Bergenti, E. Franchi and A. Poggi. Selected Models for Agent-based Simulation of Social Networks. In Proc. of the 3rd Symposium on Social Networks and Multiagent Systems (SNAMAS ‘11), York, 2011, 27--32.

[25] P. Erdös and A. Rényi. On random graphs. Publicationes Mathematicae, 6(26):290–297, 1959.

[26] M. J. North, T. R. Howe, N. T. Collier and J. R. Vos. A Declarative Model Assembly Infrastructure for Verification and Validation. In Proc. of the Advancing Social Simulation: The First World Congress, 2007, 129--140.

[27] N. Minar, R. Burkhart, C. Langton and M. Askenazi. The Swarm simulation system: a toolkit for building multi-agent simulations. Working Paper 96-06-0421996.

[28] S. Tisue and U. Wilensky. NetLogo: A simple environment for modeling complexity. In Proc. of the International Conference on Complex Systems, Boston, 2004, 16--21.

[29] S. Luke. MASON: A Multiagent Simulation Environment. Simulation, 81(7):517–527, 2005.

[30] M. Kumar, B. A. Shirazi, S. L. Das, B. Y. Sung, D. Levine, and M. Singhal. PICO: A Middleware Framework for Pervasive Computing. IEEE Pervasive Computing, 2(3):72-79, 2003.

[31] C. Becker, M. Hante, G. Schiele and K. Rotheemel. PCOM - a component system for pervasive computing. In Proc. of the 2nd IEEE Conf. on Pervasive Computing and Communications (PerCom 2004), Orlando, FL, 67-76, 2004.