AGENT CAPABILITY: AUTOMATING THE DESIGN TO CODE PROCESS

Loris Penserini [a] Anna Perini [a] Angelo Susi [a] John Mylopoulos [b]

[a] *ITC-irst, Via Sommarive 18 I-38050, Trento, Italy, {penserini,perini,susi}@itc.it*
[b] *Univ. of Trento, Via Sommarive, 14, I-38050, Trento, Italy*

Abstract

Current IT application domains such as web services and autonomic computing call for highly flexible systems, able to automatically adapt to changing operational environments as well as to user needs. We are conducting research on how to build these complex systems in the context of agent-oriented software. This short paper gives an overview of our approach (described in details in [3]) which rests on a tool-supported, development process, and aims at linking stakeholder needs, elicited during domain analysis, to agent capabilities coded into a MAS.

1 Approach Overview

Developing distributed software systems that operate in open, evolving and heterogeneous environments is becoming a critical issue in IT application domains such as web services and autonomic computing. Multi-Agent System (MAS) are offering both technological solutions as well as an effective paradigm for the development of this type of complex systems. Agents behavior at run time is influenced by the environment within which they operate [6], so designing agents able to behave in an effective way should benefit from a deep knowledge of that environment, including the stakeholders who have specific needs and expectations from the agents being designed. The specification of agents behavior is typically addressed during detail design, a step which precedes implementation. Analysis of the environment, on the other hand, is performed earlier, during domain and system requirements modelling, which rest on environment- and problem-oriented abstractions.

Our goal is to address the problem of supporting traceability between requirements, design and code artifacts, borrowing ideas and standards proposed by the Model Driven Architecture (MDA) initiative [2]. In particular, we conceives system development in terms of a chain of model transformations, namely, from a domain and stakeholder requirements model (Computationally Independent Model CIM, in MDA terminology) to a detailed design models of agents behavior, which provides a Platform Independent Model (PIM), and from a PIM to a Platform Specific Model (PSM), from which code and other development artifacts can then be straightforwardly derived.

To this aim, we extended the *Tropos* [1] agent-oriented methodology by introducing agent capability modeling since requirements analysis [4], and provided a tool-supported process. The *Tropos* methodology rests on a conceptual framework which includes the notions of *agent*, *goal*, *plan* and *social dependency* between agents for goal achievement. These knowledge level concepts are uniformly used throughout the software development process which is organized along five main phases: *Early Requirements*, *Late Requirements*, *Architectural Design*, *Detailed Design* and *Implementation*. We adopted a revised definition of the agent capability notion which distinguishes the concept of *ability* from the concept of *opportunity* [4]. The *ability* part represents a way to achieve a given goal. The *opportunity* part represents user preferences and environmental conditions, which may enable or disable the execution of the ability part, at run time.

An high level view of the development process we are proposing can be summarized as follows. **Step 1.** Capability modeling starts during requirements analysis by identifying agent capabilities as a way to accomplish stakeholder needs (CIM model). Ability part of capabilities are specified in *Tropos* by a means-end relationship between the goal and the plan. Goal/plan OR-decomposition allows to describe alternative capabilities that can achieve a goal. The *opportunity* part is described via plan/softgoal contributions (e.g. see Fig. 1, (a)), $\langle plan, softgoal, metric \rangle$

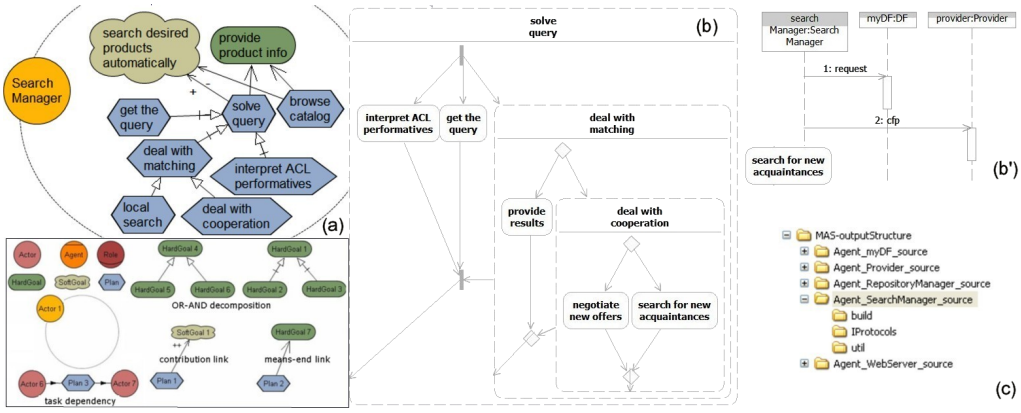

Figure 1: Fragments of development artifacts: (a) *Tropos* goal model of the *Search Manager* actor; (b) and (b') UML Detailed Design specification of a capability; (c) MAS generated structure.

($metric \in \{+, -, ++, --\}$) and domain constraints as model annotations.

Step 2. Further refinements of capability models are provided during architectural and detailed design where capability dynamic properties are specified by a set of UML 2.0 activity and sequence diagrams. These requirements analysis and design phases are supported by a *Tropos* modeler, based on the Eclipse platform, which uses a MOF [2] compliant implementation of the *Tropos* language metamodel and of the UML metamodel.

Step 3. Agent code generation results from a transformation of these PIM specifications to the target platform specific model (PSM). PIM to PSM transformation is implemented using automatic transformation techniques: for the ability part [4] while, for the opportunity part, a two steps process is adopted. The first step consists in a mapping from *Tropos* specification to Jadex, through the Jadex/JADE adapter [5], the second step goes from UML specification of capabilities to JADE. The output of this process is a skeleton of the code of JADE agents which execute plans, correspondingly to the ability specification. The choice of which plan to execute is performed by the agent according to BDI mechanisms which implement the opportunity specification.

Basic features of the resulting MAS implementation, are: **a)** each agent can play different roles according to special request-messages, related to target goals. That is, an agent can sense the environment and consequently switch to a specific role, execute the most appropriate capability, chosen by using the knowledge coded into the opportunity part; **b)** the ability part of a capability is implemented as a specialization of the class *jade.core.behaviours.FSMBehaviour*, namely it represents a final states machine (automaton). This allows to exploit monitoring mechanisms during capability execution with the aim of making the agent aware of failures and able to react to them.

References

[1] P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, and A. Perini. Tropos: An Agent-Oriented Software Development Methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, July 2004.

[2] S. J. Mellor, K. Scott, A. Uhl, and D. Weise. *MDA Distilled*. Addison-Wesley, 2004.

[3] L. Penserini, A. Perini, A. Susi, and J. Mylopoulos. From Capability Specification to Code for Multi-Agent Software. In *Proceedings of the 21st Conference IEEE/ACM International Conference on Automated Software Engineering (ASE'06)*. IEEE Press, 2006.

[4] L. Penserini, A. Perini, A. Susi, and J. Mylopoulos. From Stakeholder Intentions to Software Agent Implementations. In *Proceedings of the 18th Conference On Advanced Information Systems Engineering (CAiSE'06)*. LNCS, Springer-Verlag, 2006.

[5] A. Pokahr, L. Braubach, and W. Lamersdorf. Jadex: A BDI reasoning engine. In R. Bordini et al., editor, *Multi-Agent Programming*, 2005.

[6] M. Wooldridge. *An Introduction to Multiagent Systems*. John Wiley & Sons, 2001.