A UML Based Approach for Modeling and Implementing Multi-Agent Systems

Viviane Torres da Silva Ricardo Choren Noya Carlos José Pereira de Lucena

PUC-Rio, Computer Science Department, SoC+Agent Group,
Rua Marques de São Vicente, 225 - 22453-900, Rio de Janeiro, RJ, Brazil
{viviane, choren, lucena}@inf.puc-rio.br

PUC-RioInf.MCC 51/03 November, 2003

Abstract. In this paper we present an agent-oriented modeling language called MAS-ML and an approach for mapping MAS-ML diagrams into Java implementations. MAS-ML extends the UML meta-model describing new meta-classes and stereotypes, extending the class and sequence diagrams and proposing two new diagrams: organization and role diagram. The paper also relates MAS-ML to other modeling languages that also extend the UML for modeling multi-agent systems.

Keywords. Multi-Agent Systems, Object-oriented Systems, Modeling Languages, Programming Languages,

Resumo. Neste artigo iremos apresentar uma linguagem de modelagem chamada MAS-ML e uma abordagem para mapear diagramas MAS-ML em implementações Java. MAS-ML estende o meta-modelo UML descrevendo novas meta-classes e novos estereótipos, estendendo os diagramas de classe e seqüência e propondo dois novos diagramas: diagramas de organizações e de papéis. O artigo também relaciona MAS-ML com outras linguagens de modelagem que também estendem UML para modelagem de sistemas multi-agentes.

Palavras-chave. Sistemas Multi-Agentes, Sistemas Orientados a objetos, Linguagens de Modelagem, Linguagens de Programação.

1 Introduction

In recent years agents have become a powerful technology that can be applied to several significant applications. The development of appropriate methodologies to develop agent-based solutions is a key issue in getting the agent technology into the mainstream of software development.

A modeling language is an indispensable element in agent-based software technology. There already are several proposals for an agent modeling language such as [1,10]. Although all modeling languages define similar agent features, a standardized way does not exist yet for specifying an agent system, as there is for the object technology, the UML. Even worse, instead of using a generic programming language, the significant problem of implementing an agent system is only solved for certain specific agent architectures or models.

In an ideal setting, a developer would be able to specify an agent system without having to think about the best modeling language available for a specific architecture in which the system will be implemented. An example is the Tropos methodology [13], whose detailed design is oriented very specifically towards JACK [7] as an implementation platform. It is important to develop a modeling language that could have a universal usage, as UML does in the object-oriented world, and that could help developers generate code for a programming language.

To solve these problems, we have developed the MAS-ML (Multi-Agent System Modeling Language) [16,17], a UML [20] extension based on the TAO (Taming Agents and Objects) conceptual framework (meta-model) [15]. We also propose a mechanism to map MAS-ML diagrams to Java implementations.

TAO defines the static aspects and dynamic aspects of MAS. The static aspect of TAO captures the MASs' elements, their properties and their relationships. The elements defined in TAO are agents, objects, organizations, environments, agent roles and object roles. The relationships that link these elements are inhabit, ownership, play, control, dependency, associations, aggregation and specialization relationships. The dynamic aspects of TAO are directly related to the relationships between the elements of MAS. The dynamic aspects are domain-independent behaviors associated with the interaction between MAS elements. For instance, the creation and destruction of MAS elements and the migration of an agent from one environment to another are described as domain-independent dynamic aspects of MAS.

While extending the UML meta-model, new meta-classes and stereotypes were created to represent MAS elements and their properties. Although we have extended UML by creating new meta-classes, we have produced a conservative extension [18]; i.e., the extensions made to UML were strictly additional to the standard UML semantics, which remained unaltered. MAS-ML defines structural and dynamic diagrams to model agent-based systems. These models build a representation of the system and all of its features and provide the required functionality to generate implementation code.

The remainder of the paper is structured as follows. Section 2 describes the MAS-ML meta-model; indicating the new abstractions that were added to the UML meta-model so that a system can be modeled using the agent paradigm. Section 3 presents the MAS-ML structural diagrams, while Section 4 presents the MAS-ML dynamic diagram. Section 5 describes a tool that transforms MAS-ML structural diagrams into Java code. Section 6 describes some related work and, finally, Section 7 concludes.

2 The MAS-ML Meta-model

To provide a UML extension where agents, organizations, environments, agent roles and object roles can be represented, new meta-classes need to be created. Stereotypes were not used to define these elements because objects and these elements have different properties and different relationships. Stereotypes extend the modeling capabilities extending the semantics but not the structure of existing meta-classes [9].

The *AgentClass* meta-class was created to represent agents. This is necessary because agents are expressed through mental components as goals, beliefs, actions and plans. The *AgentClass* meta-class extends the *Classifier* meta-class, which is associated with the *StructuralFeatures* and *BehavioralFeatures* meta-classes. Goals and beliefs are defined with *Goal* and *Belief* stereotypes on the *Property* meta-class, instantiated as attributes. An action is represented through the *AgentAction* meta-class. Plans are represented through the *AgentPlan* meta-class, a specialization of the *BehavioralFeature* meta-class.

An organization, as defined in TAO, is an extension of agent. So, organizations also have goals, beliefs, actions and plans. Nevertheless, they also define axioms and roles, which can be played by the elements that inhabit them. To represent organizations, the *OrganizationClass* meta-class, an extension of the *AgentClass* meta-class, was created. An axiom is represented by the *Axiom* stereotype on the *Property* meta-class.

TAO defines two types of roles: agent and object roles. Agent roles are based on goals, beliefs, duties, rights and protocols. To represent an agent role the *AgentRoleClass* meta-class, an extension of the *Classifier* meta-class, was created. Duties and rights are represented by the *Duty* and *Right* stereotypes on the *AgentAction* meta-class. Protocols are represented by the *AgentProtocol* meta-class, which is an extension of the *BehavioralFeature* meta-class.

Since the definitions of object role in TAO and in UML are different, the *ObjectRoleClass* meta-class was created. The *ObjectRoleClass* meta-class is a specialization of the *Classifier* meta-class.

An environment, in TAO, can be modeled as an object or an agent, depending on its characteristics. So, an environment is represented as a stereotype in both the *Class* and the *AgentClass* meta-classes.

In order to be able to represent instances of these new meta-classes and stereotypes as illustrated in Figure 1, MAS-ML creates new model elements. Each element described above (agent, organization, agent role, object role and environment) has a new model element. In MAS-ML, every model element has three compartments: the top, which holds the element name; the middle, which holds the element structural

features; and the bottom, which holds the element behavioral features. For instance, the model element of an agent shows the agent name in the top compartment, its goals and beliefs in the middle one, and its actions and plans in the bottom one. Moreover, new model elements were introduced to represent all the relationships that were described in TAO but are not described in UML, such as inhabit, ownership, play and control.

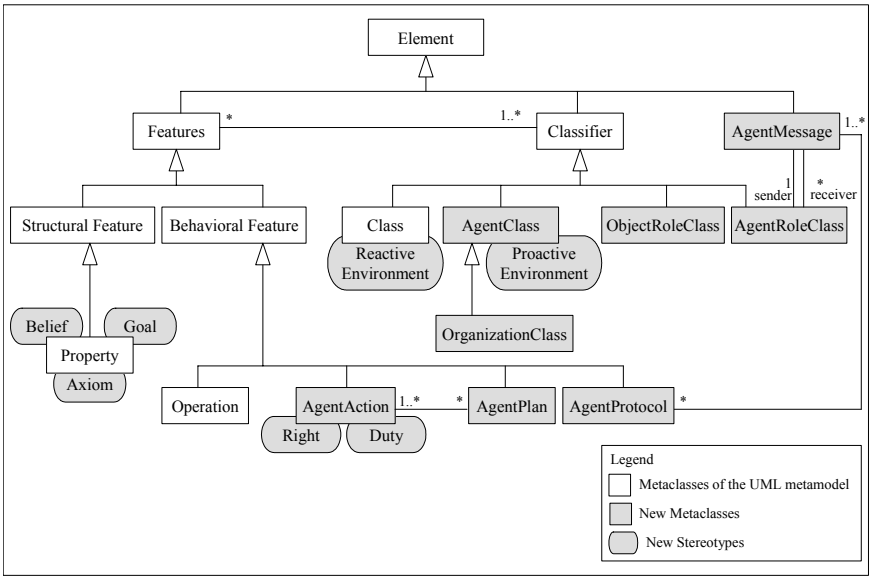

Figure 1 - MAS-ML Meta-model

3 MAS-ML Static Diagrams

MAS-ML extends the UML class diagram and defines two new diagrams to model MAS structural / static aspects.

3.1 Class Diagram

MAS-ML extends the UML class diagram to represent the structural relationships between agents, agents and classes, organizations, organizations and classes, and environments. It is not necessary to represent the relationships between environments and any other element since all the elements in the system inhabit the environment. In addition, there are only two different relationships that link two agents or two organizations – aggregation or specialization relationships. Other relationships described in TAO link agents and organizations through the roles that they play and these relationships are modeled in the roles diagram (see Section 3.3).

In the MAS-ML class diagram there can only be two relationships between agents or between organizations: aggregation and specialization. All other relationships that

link agents and organizations take place through the roles they play. The class diagram also models the relationships between objects and agents or organizations.

The entities that can be modeled by class diagrams are agents, organizations, environments, classes and other classes defined by UML. The relationships used in this diagram are those defined by UML to link classes of objects plus the association relationship (linking agents and classes, organizations and classes, or environments), aggregation relationship (linking agents, organizations or environments) and specialization relationship (linking agents, organizations or environments).

3.2 Organization Diagram

The purpose of the organization diagram (Figure 2) is to model the system organizations and the relationships between them and other system elements. This diagram models the environment that the organization inhabits, the roles defined by the organization and the objects, agents and sub-organizations that play those roles.

The entities modeled in organization diagrams are organizations, agents, agent roles, object roles and environments. The relationships used in this diagram are the ownership relationship (linking the organization and the role that it defines), play relationship (linking the roles and the agents, sub-organizations or objects that play the roles) and inhabit relationship (linking the environment and the organization).

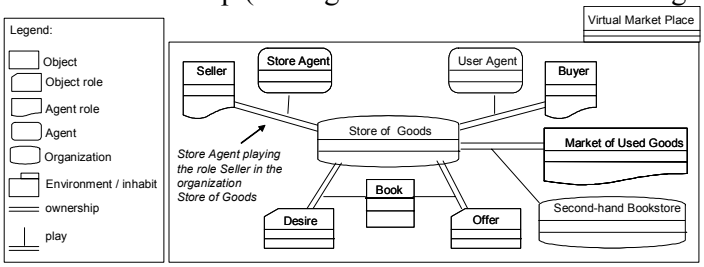

Figure 2 - Organization diagram

3.3 Role Diagram

The role diagram (Figure 3) is responsible for modeling the relationships between the roles defined in the organizations. This diagram defines the relationships between the agent roles, between agent roles and object roles, between object roles and between the roles and the classes that they use/define.

This diagram can be composed of agent roles, object roles and classes. The relationships used in this diagram are control, dependency, association, aggregation and specialization relationships.

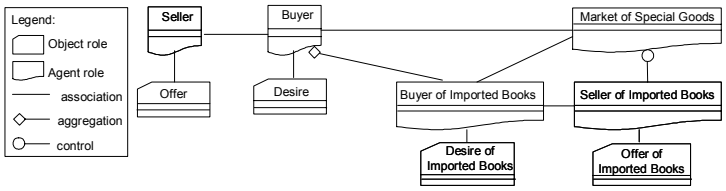

Figure 3 – Role diagram

4 MAS-ML Dynamic Diagram

A UML sequence diagram presents a set of interactions between objects playing roles in collaborations. MAS-ML extends the sequence diagram to represent the interaction between agents, organizations and environments. The extensions proposed to the UML sequence diagram were based on the domain-independent dynamic processes described in [17].

While extending the UML sequence diagram, three elements were created to represent agents, and the existing object element was modified. New pathnames describing an element in a sequence diagram were also created and the pathname that describes an object was modified in order to reflect the relationship between the object and the environment it inhabits, for instance.

MAS-ML also introduces new stereotypes (Figure 4) to identify new interaction types and has extended the definition of the stereotype <<create>> and <<destroy>> defined in the UML meta-model. The stereotype <<create>> was specialized to represent the creation of agents, organizations and environments. Since TAO defines that every agent (or sub-organization) plays at least one role, this stereotype also is used to represent the association of a role instance to an agent (or sub-organization). The stereotype <<*destroy*>> was specialized to represent the destruction of agents, organizations and environments and the destruction of all role instances associated with agents, sub-organizations and objects, since a role instance cannot exist without an element to play it.

The stereotypes <<role_commitment>>, <<role_cancel>>, <role_deactivate>>, <<role_activate>> and <<role_change>> were created to represent an element (agent, organization or object) committing to, canceling, deactivating, activating and changing a role. The stereotype <<role_cancel>> may also illustrate an agent or an organization leaving an organization. Using the stereotype <<role_commitment>>, the designer can represent an agent or an organization entering an organization to play a role. However, it is not possible to represent an agent or an organization entering a new environment because an element cannot play roles in different environments. To represent an element moving to another environment it is necessary to use the stereotype <<role_change>>.

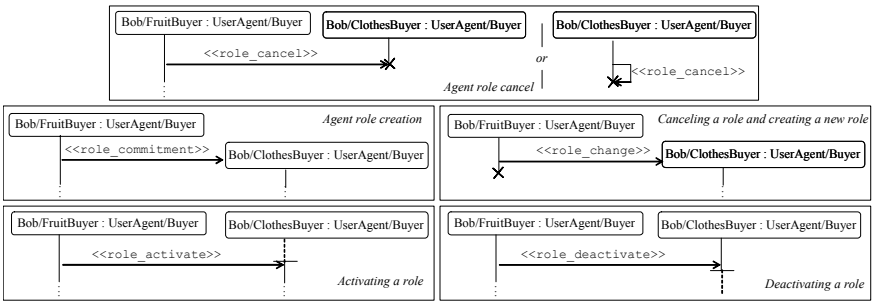

Figure 4 – Using stereotypes in Sequence diagrams

5 Generating Code from MAS-ML models

To implement a system modeled using MAS-ML, it is necessary to refine the models and to generate code. Of course, the process of generating code from agent-oriented models would be facilitated if a programming language that also considers these elements as first order abstractions was used since model elements would be directly mapped into implementation abstractions. However, no well known and widely used programming language with such a characteristic exists.

So, a transformer was developed to build Java code from the information described in MAS-ML structural diagrams. However, MAS-ML structural diagrams only describe the elements (meta-classes instances of the MAS-ML meta-model), their properties and their relationships. These diagrams do not describe the information about the system execution, which is presented in the dynamic diagram. Therefore, the code generated by the transformer is not completely executable.

The MAS-ML2Java transformer was developed using TXL [19], a programming language designed to support transformational programming. Basically, TXL transforms an input text, described according to a specified grammar, to an output text using a set of transformation rules. To transform MAS-ML structural models into Java, it was necessary to create a grammar to define the input from a MAS-ML model and a set of rules to transform this input into Java classes. For instance, this grammar describes the agents defined in the applications and the beliefs, goals, actions and plans associated with the agent.

The process defined to transform MAS-ML models into Java is composed of three phases: domain-independent phase, domain-dependent phase and relationship phase. The domain-independent phase generates a set of classes that are the basis for any MAS application. Some of these classes are abstract classes representing elements found in any MAS, such as agents and agent roles. These classes define a set of attributes and methods that are available to their sub-classes and other methods that must be implemented by the sub-classes. Other classes generated in this phase represent the properties of the elements. Depending on the characteristic of the property, they also are represented by abstract classes. For instance, an abstract class called *Agent* is created to represent abstract agents. Moreover, actions and plans are

transformed into abstract classes *Action* and *Plan* since they must be specialized according to specific actions and plans defined by each application agent.

The domain-dependent phase creates a set of classes that are generated according to the application domain. Some of these classes extend other classes generated in the domain-independent phase. For instance, suppose an application defines a user agent that has a plan called "negotiating" and an action called "evaluate proposal." A class called *User_Agent* extending the abstract class *Agent* is created and classes called *Negotiating* and *Evaluate_Proposal* are created, extending the abstract classes *Plan* and *Actions*, respectively.

The relationship phase modifies the generated classes according to the relationships defined in the application by creating attributes to represent the relationships. For instance, suppose that the agent *User_Agent* is associated with the class *Product*. Attributes to represent the association between the agent and the class are defined in this phase.

6 Related Work

Several MAS modeling languages based on the UML extension have been proposed. AUML [1,2,6,10] proposed by FIPA and OMG is the most popular one. AUML extends the UML class diagram to represent agents. In AUML, an agent is defined as a stereotype of an object. Nevertheless, agents and objects are presented as different paradigms. Stereotypes may be used to indicate a difference in meaning or usage between two model elements with identical structures and so, based on the definition presented in the UML specification, stereotypes may not be used to represent two completely different paradigms. Moreover, organizations and environments are not defined in AUML and therefore the relationships between agents and these elements cannot be modeled.

AUML also proposes to extend the UML sequence diagram to represent agent protocols. Although they define that agents can play more than one role during the execution of their protocols, they do not illustrate the representation of an agent committing to a new role or canceling one of its roles. Moreover, although during the execution of an interaction protocols agents may need to interact with organizations and the environment, these interactions are not illustrated in the AUML sequence diagram.

The main differences between MAS-ML and AUML are: (i) in MAS-ML each element is defined as a different meta-class since they define different properties and are linked by different relationships; (ii) using the static diagrams proposed by MAS-ML it is possible to model all the MAS elements defined in TAO and the relationships between them; and (iii) using the MAS-ML dynamic diagrams it is possible to model the interactions between all the MAS elements.

In [21], Wagner proposes the AORUML (Agent-Object-Relationship UML). Although agents, objects and their relationships are defined, AORUML does not describe the relationships among agents, objects and other MAS entities. Thus, it is not possible to model all the elements of the MAS application and their relationships.

Some MAS methodologies such as [3,13] also propose to use or extend UML to model MAS concepts. Tropos [13] extends the Use Case diagram based on features defined in their methodologies. Message [3] creates specific diagrams to represent agents and other MAS elements by adding those diagrams to the set of UML diagrams and not extending any UML diagram. Although there are similarities between MAS-ML and Message, none of their proposed diagrams represent the interaction among the MASs' elements.

Extensions to the UML sequence diagram also are proposed in [4,11]. In [11], the authors describe some temporal aspects of the dynamic role assignment. Their approach is very similar to MAS-ML since it describes operations associated with roles (agents committing to new roles, agents activating roles, agents deactivating roles, agents canceling roles and agents changing roles) and it presents a graphical representation of these operations, extending the UML sequence diagram. In [4], the authors propose a sequence diagram called an organization sequence diagram to represent agents playing roles in a specific group. Although it is possible to represent an agent playing different roles in different organization, it is not possible to represent the interactions between organizations and agents. Agents need to request their participation in organizations to play one of the available roles.

7 Conclusions and Ongoing Work

The main contributions of our work can be summarized in two aspects: (i) a modeling language that extends UML based on a conceptual framework; and (ii) the mapping of the agent-level design elements to an object-oriented programming language.

Using the MAS-ML meta-model and diagrams, it is possible to represent the elements associated with a MAS application and to describe the static relationships and interactions between these elements. The UML meta-model was extended to include all agent-oriented concepts defined in TAO. New meta-classes, stereotypes and relationships were added to the UML. Moreover, new diagram elements and new diagrams were proposed to represent the elements, the relationships and the interactions between the elements. The extended class diagram models all the classes, some agents and some organizations defined by the application. The organization diagrams identify all the organizations of the application and consequently all the agent roles and object roles. Since every agent and sub-organization play at least one role and the organization diagrams model the elements that play the roles, all agents and all sub-organizations are defined in these diagrams. The role diagram models all the relationships between the roles of the system and between the roles and the classes. Using the extended sequence diagram it is possible to model (i) the creation and destruction of agents, organizations, roles and environments (primitive dynamic aspects), (ii) agents and organizations committing to roles, changing roles, activating and deactivating their roles and (iii) agents and organizations changing their habitat and changing from one organization to another.

The MAS-ML2Java transforms MAS-ML structural diagrams into Java code. All elements and relationships presented in the structural diagrams are mapped into Java

classes according to the characteristics of the elements. In order to improve the generated code, the authors intend to analyze agent-oriented platforms and architectures, programming languages and patterns such as [5,8,14]. Our aim is to evaluate the mapping of agent-oriented design models and concepts to object-oriented and agent-oriented programming languages.

It may also be necessary to improve the MAS-ML sequence diagram to generate code from the system dynamic diagrams. The authors believe that the sequence diagram should be extended to clearly represent the selection of plans, the execution of plans and actions and the message exchanges, for instance. To extend the sequence diagram, MAS applications published as benchmarks in the literature and others developed in our group are being analyzed. Moreover, approaches that also propose an extension of the UML sequence diagram are being investigated and used in experiments. The code generated by the dynamic diagrams will implement the methods of the classes that could not be implemented during the automatic generation of code from the static diagrams.

Acknowledgement. This work has been partially supported by CNPq under grant 140646/2000-0 for Viviane Torres da Silva. Viviane Torres da Silva, Ricardo Choren and Carlos J. P. de Lucena are also supported by the PRONEX Project under grant 664213/1997-9, by the ESSMA project under grant 552068/2002-0 and by the 1st art. of decree number 3.800, of 04.20.2001.

References

1. Bauer, B.: UML Class Diagrams revisited in the context of agent-based systems. In: M. Wooldridge, P. Ciancarini, and G. Weiss (Eds.) Proceedings of Agent-Oriented Software Engineering (AOSE 01), LNCS 2222, Montreal ,Canada (2001):1-8.
2. Bauer, B., Müller, J., Odell, J.: Agent UML: A Formalism for Specifying Multiagent Interaction. In: Ciancarini, P., Wooldridge, M. (Eds.), Agent-Oriented Software Engineering, Springer-Verlag, Berlin (2001):91-103.
3. Caire, G.: MESSAGE: Methodology for Engineering Systems of Software Agents Initial Methodology. In: Technical report, EDIN 0224-0907, Project P907, EURESCOM (2001).
4. Ferber, J., Gutknecht, O., and Michel, F.: From Agents to Organizations: an Organizational View of Multi-Agent Systems. In: Proceeding of the Fourth International Workshop on Agent-Oriented Software Engineering (AOSE), Melbourne, Australia, (2003).
5. FIPA, "Foundation for Intelligent Physical Agents," http://www.fipa.org, (2003).
6. Huget, M.: Agent UML Class Diagrams Revisited. In: Bauer, B., Fischer, K., Muller, J. and Rumpe, B. (Eds.), Proceedings of Agent Technology and Software Engineering (AgeS), Erfurt, Germany, (2002).
7. Jack Intelligent Agents, http://www.agent-software.com (2003).

8. Jade: Java Agent Development Framework, http://sharon.cselt.it/projects/jade/ (2003).
9. Lind, J.: Specifying Agent Interaction Protocols with Standard UML. In: Proceedings of the Second International Workshop on Agent Oriented Software Engineering (AOSE), Montreal, Canada, (2001):136-147
10. Odell, J., Parunak, H. and Bauer, B.: Extending UML for Agents. In: Wagner, G., Lesperance, Y. and Yu, E. (Eds.), Proceedings of the Agent-Oriented Information Systems Workshop, AOIS 2000, Eds., Austin, (2000): 3-17.
11. Odell, J., Parunak, H., Bruechner, S., Fleisher, M.: Temporal Aspects of Dynamic Role Assignment. In: Giorgini, P, Müller J., Odell, J. (Eds.) Agent-oriented Software Engineering (AOSE), LNCS, Berlin, (2004) (forthcoming).
12. OMG, "Object Management Group," http://www.omg.org, (2003).
13. Mylopoulos, J., Kolp M., Castro J.: UML for Agent-Oriented Software Development: the Tropos Proposal. In Proceedings of the Fourth International Conference on the Unified Modeling Language, Toronto, Canada (2001).
14. Shoham. Y.: AGENT0: A simple agent language and its interpreter. In: Ninth National Conference on Artificial Intelligence, Anaheim, USA (1991)
15. Silva, V., Garcia, A., Brandao, A., Chavez, C., Lucena, C., Alencar, P.: Taming Agents and Objects in Software Engineering. In: Garcia, A., Lucena, C., Zamboneli, F., Omicini, A, and Castro, J., (Eds.), Software Engineering for Large-Scale Multi-Agent System, LNCS, Springer-Verlag, (2003).
16. Silva V., Lucena C.: From a Conceptual Framework for Agents and Objects to a Multi- Agent System Modeling Language. In: Technical Report CS2003-03, School of Computer Science, University of Waterloo, Canada (2003). (under revision)
17. Silva, V., Lucena, C.: Extending the UML Sequence Diagram to Model the Dynamic Aspects of Multi-Agent Systems, In: Technical Report MCC15/03, PUC-Rio. Rio de Janeiro, Brazil (2003). (under revision)
18. Turski, W., Maibaum, T.: Specification of computer programs, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, (1987)
19. TXL: The TXL programming language, Version 10.3, http://www.txl.ca, (2003).
20. UML: Unified Modeling Language Specification, Version 2.0, http://www.omg.org/uml/, (2003).
21. Wagner, G.: The Agent-Object-Relationship Metamodel: Towards a Unified View of State and Behavior. Information Systems 28:5 (2003).