PUC

ISSN 0103-9741

Monografias em Ciência da Computação

nº 10/05

Multi-Agent System Design Verification Using Knowledge-Based Reasoning

Anarosa Alves Franco Brandão
Viviane Torres da Silva
Carlos José Pereira de Lucena

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO
RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22453-900
RIO DE JANEIRO - BRASIL

Monografias em Ciência da Computação, No. 10/05 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena March, 2005

Multi-Agent System Design Verification Using Knowledge-Based Reasoning*

Anarosa Alves Franco Brandão Viviane Torres da Silva
Carlos José Pereira de Lucena

{anarosa,viviane,lucena}@inf.puc-rio.br

Abstract. This paper presents a knowledge-based approach to the verification of the design consistency of multi-agent systems modeled by using MAS-ML, which is a multi-agent system modeling language. We provide a formal description of the MAS-ML diagrams as well as a reasoning engine that are used together to detect intra-model and inter-models inconsistencies. Intra-model inconsistencies are detected based on the specification of each diagram provided by the MAS-ML formal description. Inter-models inconsistencies are identified based on the interdependencies between the diagrams described in the reasoning engine.

Keywords: Design Verification, Knowledge-Based Reasoning, Multi-Agent Systems.

Resumo: Este artigo apresenta uma abordagem baseada em conhecimento para a verificação de consistência de modelos de design de sistemas multi-agentes. Os modelos de design considerados são descritos em MAS-ML, uma linguagem de modelagem para sistemas multi-agentes. A verificação é feita a partir da descrição formal dos diagramas de MAS-ML a fim de utilizarmos um conjunto de regras de inferência para detectar inconsistências intra-modelos e inter-modelos. Inconsistências intra-modelos são identificadas a partir da especificação formal de cada diagrama MAS-ML. Inconsistências inter-modelos são identificadas a partir da definição formal das interdepend6encias entre os diagramas.

Palavras-chave: Verificação de Modelos de Design, Raciocínio Baseado em Conhecimento, Sistemas Multi-Agentes.

* This work has been sponsored by the Ministério de Ciência e Tecnologia da Presidência da República Federativa do Brasil, CNPq/Brazil, under the Project ESSMA number 5520681/2002-0

In charge for publications:
Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC-Rio Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22453-900 Rio de Janeiro RJ Brasil
Tel. +55 21 3114-1516 Fax: +55 21 3114-1530
E-mail: bib-di@inf.puc-rio.br

1 Introduction

Multi-agent systems are gaining wide acceptance in both industry and academia as a powerful paradigm for developing software systems. However, the development of multi-agent systems (MAS) is more complex than the development of object-oriented systems, due to the intrinsic characteristics of such systems [6, 14]. MAS are composed not only by objects but also by agents that inhabit environments and may play roles in several organizations [9]. These new abstractions (agents, organizations, environments and roles), their structural and dynamic properties together with their relationships, make the modeling and implementation of MAS very difficult tasks.

Numerous MAS modeling languages have been proposed in the literature [13, 8, 10] in order to help the designers to model the many facets of the systems. These languages are usually more complex than object-oriented languages, such as UML, since they frequently provide support for modeling almost all the MAS abstractions and their characteristics by using structural and dynamic diagrams.

The different abstractions and relationships modeled by using any of the diagrams proposed by a MAS modeling language with the interdependencies between the diagrams contribute to generate intra-model and inter-models inconsistencies. Intra-model inconsistencies are identified by analyzing the specification of each diagram. The specification of a diagram defines it as a structural or dynamic diagram and describes what can be modeled by using the forementioned diagram.

Inter-model inconsistencies result from the interdependencies between the diagrams. Different diagrams model different aspects (or views) of the MAS and, therefore, different aspects of the same entity are usually modeled in several diagrams. Thus, the overall MAS modeling may become inconsistent, although each model is consistent itself.

In this paper we propose the use of a DL (description logics) [1] knowledge-based reasoner [19] to verify the consistency of MAS-ML designs [10, 11]. MAS-ML is a MAS modeling language that provides structural and dynamic diagrams to model MAS abstractions. This paper focus on detecting the structural inconsistencies, i.e., the inconsistencies that may occur in each structural diagrams and between the three structural diagrams proposed by MAS-ML. Our approach provides a formal description of the MAS-ML modeling language as well as a reasoning engine that is used to detect inconsistencies between models. The MAS-ML formal description is based on the MAS-ML metamodel [10] that provides the specification of the diagrams which compose the modeling language.

This paper is organized as follows. Section 2 briefly presents the structural diagrams of the MAS-ML modeling language. Section 3 summarizes the MAS-ML formal model defined to check the intra-model consistencies of the MAS-ML structural diagrams. Section 4 partially presents the reasoning engine used to verify the inter-model consistencies. Section 5 introduces some related work, and finally, Section 6 concludes and presents the ongoing work.

2 The MAS-ML Static Diagrams

MAS-ML is a modeling language that extends UML in a conservative way to describe the MAS specificities that cannot be described by using UML. MAS-ML defines a set of structural diagrams that can be used to describe the structural properties of the MAS abstractions and their relationships. In this sense, the UML class diagram was extended to accommodate the representation of the new abstractions and two other new structural diagrams (organization and role diagrams) where described to focus on complementary viewpoints.

2.0.1 Class Diagram

The extended class diagram proposed by MAS-ML focus on the representation of agent classes, organization classes, environment classes and the relationships between these abstractions and object classes. The relationships that can be used in this diagram are *inhabit* (relating the object classes to the environment classes they inhabit), *specialization* (relating the specialization between entities of the same class type) and *association* (describing the associations between the MAS entity classes and object classes). Since the MAS-ML class diagram is a conservative extension of the UML class diagram, every abstraction and relationship that can be modeled by using the UML class diagram can also be modeled by using the extended class diagram.

2.0.2 Organization Diagram

Organization diagrams model the system organization classes, their properties, the role classes that they define, the entity classes that play these roles and the environment classes in which they inhabit. Each organization diagram focus on modeling the characteristics of one organization. The organization diagram is the richest MAS-ML diagram since all the classes defined in the MAS-ML metamodel participate in this. The relationships that can be used in this diagram are *ownership* (relating the organization class being modeled to the role classes that it defines), *play* (relating agent, organization and object classes to the role classes that they play) and *inhabit* (relating the agent and organization classes to the environment classes they inhabit).

2.0.3 Role Diagram

The role diagrams focus on modeling the relationship between the role classes identified in the organization diagrams and between the role classes and the resources (or object classes) available in the organizations. This diagram shows the relationships *control* (relating roles classes that control other role classes), *dependency* (relating a role class to another role class that depends on it), *aggregation* (relating the role classes that aggregate other role classes), *specialization* (relating role classes to the role classes that they specialize) and *association* (relating two role classes by associating them).

3 Intra-model Consistency

In order to proceed with the verification of the intra-model consistency, we developed an ontology based on the MAS-ML metamodel. The purpose of this ontology is to formally describe MAS-ML models in order to allow their verification. Thus, the intra-model consistency is verified based on the ontology concepts, properties and axioms. We are using DL to describe our ontology and the RACER [19] system to check its consistency and to reason about it.

The MAS-ML diagrams (class, role and organization diagrams), MAS-ML abstractions (for instance, `agent-class` and `organization-class`), the entities modeled in MAS-ML diagrams (for example, *User Agent Class* which is an instance of the MAS-ML abstraction `agent-class`) and the MAS-ML relationships are defined as ontology concepts. The ontology properties are used to connect abstractions and relationships or to associate abstractions with their intrinsic properties (for instance, `agent-class` and the `has-goal` property). The ontology properties also connect the diagrams to the abstractions and relationships that can be used in each diagram. The ontology axioms define the ontology taxonomy and the semantics of its concepts and properties.

To describe the MAS-ML static diagrams, we define the concepts `class-model`, `organization-model` and `role-model`. The properties `has-class` and `has-relationship` are used to define the classes and the relationships that can be modeled in each diagram. Such properties are identified while defining the semantics of each diagram in the axioms. Description 1 represents a fragment of the ontology taxonomy which shows the MAS-ML models.

$$\begin{array}{lcl} \mathit{class\text{–}model} & \sqsubseteq & \mathit{model} \\ \mathit{organization\text{–}model} & \sqsubseteq & \mathit{model} \\ \mathit{role\text{–}model} & \sqsubseteq & \mathit{model}. \end{array} \tag{1}$$

According to the definition of organization diagrams presented in Section 2.0.2, and considering that the `class` concept is a general concept that represents the disjoint union of the concepts `agent-class`, `organization-class`, `object-class`, `agent-role-class`, `object-role-class` and `environment-class`, we can formally describe part of its semantics as follows:

$$\begin{array}{rcl} \mathit{organization\text{–}model} & \sqsubseteq & ((\forall\ \mathit{has\text{–}class}\ \mathit{class}) \sqcup \\ & & (\exists\ \mathit{has\text{–}relationship}\ \mathit{ownership}) \\ & & \sqcup\ (\exists\ \mathit{has\text{–}relationship}\ \mathit{play}) \\ & & \sqcup\ (\exists\ \mathit{has\text{–}relationship}\ \mathit{inhabit}). \end{array} \tag{2}$$

Ontology concepts such as `agent-class` and `organization-class` are used to define the MAS-ML abstractions agent class and organization class and the concepts `agent` and `organization` are used to represent the entities modeled in MAS-ML diagrams that are instances of the MAS-ML abstractions. The two concept types are connected by the property `is-instanceOf`. These definitions allow us to instantiate and to verify all MAS-ML models. Description 3 exemplifies the use of the property `is-instanceOf`.

$$\begin{array}{lcl} \mathit{agent} & \sqsubseteq & \forall\ \mathit{is\text{–}instanceOf}\ \mathit{agent\text{–}class} \\ \mathit{organization} & \sqsubseteq & \forall\ \mathit{is\text{–}instanceOf organization\text{–}class} \end{array} \tag{3}$$

In order to describe the intrinsic properties of MAS-ML abstractions, such as *goals* and *beliefs*, we define the ontology properties `has-goal` and `has-belief`, among others.

The semantics of entities and their properties are defined in the axioms. For instance, an agent is defined as an entity which, among other things, has at least one goal. description 4 defines such restriction.

$$\mathit{agent\text{–}class} \sqsubseteq \forall\ \mathit{has\text{–}goal}\ \mathit{goal} \tag{4}$$

The two ontology properties `has-end` and `is-end` are used to connect the MAS-ML abstractions and the relationships. The first property relates the relationships to the abstractions while the second property does the reverse. Therefore, the `has-end` property is an inverse property of `is-end`. Their definition is presented in description 5.

$$\begin{array}{rcl} \exists\ \mathit{has\text{–}end}\top & \sqsubseteq & \mathit{relationship} \\ \top & \sqsubseteq & \forall\ \mathit{has\text{–}end}\ \mathit{class} \\ \exists\ \mathit{is\text{–}end}\top & \sqsubseteq & \mathit{class} \\ \top & \sqsubseteq & \forall\ \mathit{is\text{–}end}\ \mathit{relationship} \end{array} \tag{5}$$

The MAS-ML relationships were defined as ontology concepts and their semantics were defined by using the ontology axioms. For instance, `play` is a concept in the MAS ontology which represents a relationship allowed between `agent-class` and `agent-role-class`, or between `organization-class` and `agent-role-class`, or between `object-class` and `object-role-class`. Description 6 shows the axiom that presents the `play` semantics.

$$
\begin{aligned}
play \quad \sqsubseteq \quad & (((\exists has\text{–}end_1 \ agent\text{–}class) \sqcap \\
& (\forall has\text{–}end_2 \ agent\text{–}role\text{–}class)) \sqcup \\
& ((\exists has\text{–}end_1 \ organization\text{–}class) \sqcap \\
& (\forall has\text{–}end_2 \ agent\text{–}role\text{–}class)) \sqcup \\
& ((\exists has\text{–}end_1 \ object\text{–}class) \sqcap \\
& (\forall has\text{–}end_2 \ object\text{–}role\text{–}class)))
\end{aligned} \tag{6}
$$

Nevertheless, the `play` relationship has some constraints associated with the entities it relates. In fact, MAS-ML establishes that all agent plays at least one role, i.e., all agent class must be related to an agent role class through a play relationship. Such restriction is presented in description 7.

$$
agent\text{–}class \quad \sqsubseteq \quad \forall \ is\text{–}end_1 \ play. \tag{7}
$$

The following subsection presents an example of a MAS modeled by using MAS-ML as an instance of the previously defined ontology. The diagram explored by the example is the organization diagram and it is part of the knowledge-base (KB) generated by the ontology plus its instance. The fragment of the KB code illustrated in the subsection was generated in RACER according to the MAS-ML formal semantic depicted in this section.

3.1 Organization Diagram Example

Our working example is an ontology instance referring to a Virtual Marketplace domain. By virtual marketplaces we mean markets that are located in the Web and where users buy items. Each virtual marketplace is composed of a main-market (called GeneralStore) where users are able to negotiate books. Agents called UserAgent represent the users in the market. Such agents play the role Buyer whenever they want to buy an item. The buyers look for sellers and send them descriptions of the desired items. Agents called StoreAgent playing the role Seller represent the vendors of the market. The sellers are responsible for sending the price of the items to the buyers.

The KB code below partially represents the organization diagram of the system. Such system was described by using RACER. A RACER code is composed of concepts, concept instances and relations between these instances. For example, the code representing the system shows the instantiation of an organization diagram called `org1-diagram`, an organization class called `general-store`, an agent class called `user-agent`, a play relationship called `play-1`, and so on. All those classes and relationships are defined in the organization diagram `org1-diagram`.

The code also shows the entities linked by each relationship and the properties associated with them. For example, the relationship instance `play-1` links `user-agent` (related through the `has-end1` property) and `buyer` (related through the `has-end2` property).

```
1.  (instance org1-diagram organization-model)
2.  (instance general-store organization-class)
3.  (instance user-agent agent-class)
4.  (instance buyer agent-role-class)
5.  (instance virtual-market passive-environment-class)
6.  (instance book object-class)
7.  (instance desire object-role-class)
8.  (instance play-1 play)
9.  (instance own-1 ownership)
10.  (instance inhabit-1 inhabit)
11.  (instance ctrl-1 control)
```

```
12.  (related org1-diagram general-store has-class)
13.  (related org1-diagram user-agent has-class)
14.  (related org1-diagram buyer has-class)
15.  (related org1-diagram virtual-market has-class)
16.  (related org1-diagram book has-class)
17.  (related org1-diagram desire has-class)
18.  (related org1-diagram play-1 has-relationship)
19.  (related org1-diagram own-1 has-relationship)
20.  (related org1-diagram inhabit-1 has-relationship)
21.  (related org1-diagram ctrl-1 has-relationship)

22.  (related play-1 user-agent has-end1)
23.  (related play-1 buyer has-end2)
24.  (related own-1 general-store has-end1)
25.  (related own-1 buyer has-end2)
26.  (related inhabit-1 user-agent has-end1)
27.  (related inhabit-1 virtual-market has-end2)
28.  (related ctrl-1 buyer has-end1)
29.  (related ctrl-1 desire has-end2)
```

3.2 Intra-model Verification

In order to exemplify the verification of the KB code presented in Section 3.1, we point out two intra-model inconsistencies. By analyzing the KB code, we notice that the concept `goal` is not instantiated and, consequently, the relationship between instances of the concepts `agent-class` and `goal` is not defined. Since the Description 4 specifies that every agent must have at least one goal, we found an intra-model inconsistency. To overcome this problem, we need to add the two lines described below to the code in Section 3.1.

```
30.  (instance buy-a-book goal)
31.  (related user-agent buy-a-book has-goal)
```

Another intra-model inconsistency is related to the `control` relationship. Description 2 states that the control relationship is not a relationship that can be modeled in an organization diagram. The lines `21`, `28` and `29` insert an intra-model inconsistency to the organization diagram description. In order to resolve such inconsistency, the lines should be removed.

4 Inter-models Consistency

Although each structural model can be consistent itself, the overall MAS modeling may become inconsistent due to the interdependencies between the MAS-ML diagrams. A set of rules was defined in the MAS-ML ontology describing interdependencies between the diagrams. Therefore, a reasoning engine is applied to these rules to reason about the inter-model consistency. Several inter-model inconsistencies can be generated by grouping the models together. In this paper we describe two different rules that are used to verify two different types of inter-model consistency.

Since the same entity can be modeled in several structural diagrams, it is important to ensure that the intrinsic properties of the entity are exactly the same in all diagrams. Moreover, it is also needed to check if an entity defined in a class or role diagram is defined in an organization diagram as well. The set of organization diagrams defines all

the role classes that can be played in organizations and all the agent and organization classes described in the system. Description 8 states a rule that verifies if each agent role class described in a role diagram is also defined in any of the organization diagrams. By applying the reasoning engine to this rule, the result (if positive) is composed of triples of role diagram names, agent role class names and organization class names where the agent roles are defined in the role diagrams and are not in the organization diagrams. This result helps the designer to improve the modeling quality, since it shows where one must add (or delete) a missing (or extra) class to a diagram.

```
(retrieve   (?rd ?rc ?od)
            and (?rd role-model)
                (?rc agent-role-class)
                (?od organization-model)
                (?rd ?rc has-class)
                (not (?od ?rc has-class)))
```
(8)

Another important example of inter-model consistency is associated with the entity properties. Sometimes it is important to verify if entity properties are consistent with the properties of another entity. For instance, a subset of the goals of an agent must be compatible with a subset of the goals of the roles the forementioned agent plays. An agent plays a role in order to achieve its own goals and chooses the roles that it wants to play based on the compatibility between its goals and the goals of the roles. Therefore, while defining an agent class and an agent role class related by the play relationship, the goals of these entities must be compatible. Description 9 depicts a rule that verifies if such goals are compatible. By applying the reasoning engine to this rule, the result (if positive) is a tuple composed of goal names, agent class names, agent role class names and play relationship names where the agent classes are related to agent role classes but the agent goals do not have any common goal with the agent role goals. This is another result that helps designers to improve the modeling activity quality, since it shows where there are some unwanted mismatchings between entities properties.

```
(retrieve   (?gl ?agc ?agr ?pl)
            (and
                (and (?gl goal)
                   (?agc agent-class)
                   (?agr agent-role-class)
                   (?pl play)
                   (?pl ?agc has-end1)
                   (?pl ?agr has-end2)
                   (?agc ?gl has-goal)
                )
                (not(?agr ?gl has-goal))
             )
)
```
(9)

4.1 Role Diagram Example

In order to exemplify inter-model consistencies, a fragment of the KB code describing a role diagram is presented below, by using the same example defined in Section 3.1. The RACER code of the role diagram describes two agent role classes and one object role class. The control relationship previously defined in the organization diagram was redefined in the role diagram. In the next subsection we check the inter-model consistency between the organization diagram presented in Section 3.1 and the role diagram.

```
11.  (instance ctrl-1 control)

21.  (instance role-diagram role-model)

28.  (related ctrl-1 buyer has-end1)
29.  (related ctrl-1 desire has-end2)

32.  (related role-diagram ctrl-1 has-relationship)
33.  (instance seller agent-role-class)
34.  (instance assoc-1 association)

35.  (related role-diagram seller has-class)
36.  (related role-diagram buyer has-class)
37.  (related role-diagram desire has-class)
38.  (related role-diagram assoc-1 has-relationship)
39.  (related assoc-1 buyer has-end1)
40.  (related assoc-1 seller has-end2)
```

4.2 Inter-model Verification

By analyzing the inter-model consistency between the organization and the role diagrams it is possible to notice that the agent role class called `seller`, represented in the role diagram, is not defined in the organization diagram. According to Description 8, every role class represented in a role diagram must have been defined in an organization diagram as well. In order to solve such inconsistency, it is necessary to define an `agent-class` to `play` the `seller` role in an `organization-model`, as described in the lines `41` to `47` bellow.

```
41.  (instance store-agent agent-class)
42.  (instance play-2 play)

43.  (related org1-diagram store-agent has-class)
44.  (related org1-diagram seller has-class)
45.  (related org1-diagram play-2 has-relationship)

46.  (related play-2 store-agent has-end1)
47.  (related play-2 seller has-end2)
```

Another inconsistency that can be found by checking the inter-model consistency between the role and organization diagrams is a property inconsistency. Since the role class `buyer` and the agent class `user-agent` are related by a `play` relationship, their goals must be compatible. Therefore, it is necessary to define at least one goal to the role class `buyer` that is compatible with the goals of the `user-agent`, as on the line `48`.

```
48.  (related buyer buy-a-book has-goal)
```

5 Related Work

The use of modeling languages and techniques gained popularity with the regular use of UML and similar modeling languages - not only during the analysis and design, but also in other phases of system development. However, these modeling languages do not have a precise semantics yet. Several works are addressed to tackle the problem of design models

verification [5, 7, 12, 2, 3]. Also, the pUML group [18] has the definition of a precise semantics for the UML core concepts as its main research concern.

Kim and Carrington [5] gives a translation from a UML class model to an Object-Z specification, but they don't provide means to verify the model. Our work uses an ontology based on the modeling language metamodel to provide a formal description of MAS-ML models and uses knowledge-based reasoning techniques to verify these models consistency.

Berardi [2] uses DL to formally describe a UML class diagram and the CORBA-FaCT [15] and the RACER system to reason about them in order to classify the models concerning their consistency. Such work is very similar to ours, however, our work provides means to classify not only the extended class diagram but the organization and role diagrams as well. In addition, we provide the consistency checking between MAS-ML structural models.

Mens, Straeten and Simmonds [7, 12] uses DL to detect inconsistencies and to maintain consistency between UML models in a context of software evolution. Due to the context of their work, they only consider consistency checking between different models. They define the *Classless instance* conflict [12] as the conflict that arises when an object in a sequence diagram is the instance of a class that doesn't exist in any class diagram. Our work consider a MAS context and extends the idea of *classless instance* only to *classless* when verifying the absence, in any organization diagram, of classes that were predefined in role diagrams or class diagrams.

Ekenberg and Johannesson [3] defines a logic framework for determining design correctness. Their framework is described in FOL (first order logic) and it provides guidelines to translate UML models and to detect some inconsistencies in the models. Their framework is general and the use of the translation rules depend on the designer skills in FOL, since there is not an automatic support for this activity yet. In our work we use DL, a decidable subset of FOL and, by using an ontology that describes the modeling language metamodel we turn the translation of MAS-ML diagrams to DL into an ontology instantiation. The instantiation can be made automatically by using systems such as RACER with RICE [20], Protegé 2000 [17], OilEd (with FaCT)[16], among others. Also, the verification of consistency is made automatically by checking the consistency of the KB itself (intra-model verification) or by using queries previously defined.

6 Conclusion and Future Work

Our approach could be used to check the consistence of models described by using other MAS modeling languages. The metamodel that describes the modeling language should be used to generate the ontology that formally describes the language and its diagrams. The ontology would, therefore, describe the intra-model rules and inter-models dependencies that should be used to verify the models inconsistencies.

We are in the process of extending our approach to verify the consistence between MAS-ML dynamic models and between the structural and dynamic models. Moreover, we are developing a MAS-ML tool where designers could model MAS by using the MAS-ML diagrams and could also check the models. The tool should transform the graphic models into RACER code in order to verify their consistency.

Our final goal is to automatically solve the inconsistencies, when it is possible, or to provide guidelines for the designers to solve them. We are also defining several rules to be applied to the models in order to infer if such models could be improved and to provide tips for enhancing them. Such rules are being defined based on the characteristics of MAS.

Acknowledgment

This work is partially supported by CNPq/Brazil under the project "ESSMA", number 5520681/2002-0 and the grant No. 140179/95-0 for Anarosa A. F. Brandão.

References

[1] F. BAADER, D. McGUINESS, D. NARDI and P. PATEL-SCHNEIDER (Eds) **The Description Logic Handbook: Theory, Implementation and Applications.** Cambridge University Press, 2003.

[2] D. BERARDI, Using DLs to reason on UML class diagrams, in *Proceedings of th KI-2002 Workshop on Applications of Description Logics*, 2002, available at http://ceur-ws.org/Vol-63/.

[3] L. EKENBERG and P. JOHANNESSON, A framework for determining design correctness, **Knowledge Based Systems**, Elsevier, vol , pp.1-14, 2004.

[4] V. HAARSLEV, R. MOLLER, R. STRAETEN and M. WESSEL, Extended Query facilities for Racer and an Application to Software Engineering Problems, in **Proceedings of the 2004 International Workshop on Description Logics**, pp. 148-157, 2004.

[5] S. KIM and D. CARRINGTON, A Formal Mapping Between UML Models and Object-Z Specifications, in **Proceedings of the ZB'2000, International Conference of B and Z Users, York, UK**, 2000.

[6] N. JENNINGS, On Agent-based Software Engineering **Artificial Intelligence**, vol 117, n 2, pp. 277-296, 2000.

[7] T. MENS, R. STRAETEN and J. SIMMONDS, Maintaining Consistency between UML Models with Description Logic Tools, in **Proceedings of the Workshop on Object-Oriented Reengineering at ECOOP 2003**, 2003.

[8] J. ODELL, H. PARUNAK, and B. BAUER, Extending UML for Agents. **Proceedings of the Agent-Oriented Information Systems Workshop**, Austin, pp. 317, 2000.

[9] V. SILVA, A. GARCIA, A. BRANDÃO, C. CHAVEZ, C. LUCENA, P. ALENCAR, Taming Agents and Objects in Software Engineering. In: A. Garcia, C. Lucena, F. Zambonelli, A. Omicini, J. Castro *et al* (Eds) **Lecture Notes in Computer Science**, vol 2603, pp. 1-26, Springer-Verlag, 2003.

[10] V. SILVA and C. LUCENA, From a Conceptual Framework for Agents and Objects to a Multi-Agent System Modeling Language, in K. Sycara and M. Wooldridge (Eds) **Journal of Autonomous Agents and Multi-Agent Systems**, vol 9, pp. 145-189, Kluwer Academic Publishers, 2004.

[11] V. SILVA, R. CHOREN, C. LUCENA, A UML Based Approach for Modeling and Implementing Multi-Agent Systems, **Proceeding of the third International Conference on Autonomous Agents and Multi-Agents Systems**, vol 2, pp. 914-921, July, New York, USA, 2004.

[12] R. STRAETEN and J. SIMMONDS, Detecting Inconsistencies between UML Models Using Description Logic, in **Proceedings of the 2003 International Workshop on Description Logics**, available at http://CEUR-WS.org

[13] G. WAGNER, The Agent-Object-Relationship Metamodel: Towards a Unified View of State and Behavior, **Information Systems**, vol 28, n 5, 2003.

[14] M. WOOLDRIDGE, P. CIANCARINI, Agent-Oriented Software Engineering: the State of the Art. In: P. Ciancarini, M. Wooldridge. (Eds) **Agent-Oriented Software Engineering**, LNCS 1957, Berlin: Springer, pp. 1-28, 2001.

[15] FaCT: Fast Classification of Terminologies, available at: `http://www.cs.man.ac.uk/ horrocks/FaCT` last access: Dec/12/2004

[16] OilEd, Ontology Editor, available at: `http://oiled.man.ac.uk/index.shtml` last access:Dec/12/2004

[17] Protegé 2000: Ontology Editor and Knowledge Acquisition System, available at: `http://protege.stanford.edu/` last access:Dec/12/2004

[18] pUML: The Precise UML Group, available at: `http://www.cs.york.ac.uk/puml/` last access:Dec/12/2004

[19] RACER Semantic Middleware for Industrial Projects Based on RDF/OWL, available at: `http://www.sts.tu-harburg.de/ r.f.moeller/racer/` last access:Dec/12/2004

[20] RICE: Racer Interactive Client Environment, available at: `http://www.b1g-systems.com/ronald/rice/` last access:Dec/12/2004