ISSN 0103-9741

Monografias em Ciência da Computação
n° 25/05

An Ontology-based Method for Structuring Multi-Agent Systems Formal Specifications

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. 25/05 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena Junho, 2005

An Ontology-based Method for Structuring Multi-Agent Systems Formal Specifications *

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

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

Abstract. Agents are becoming a popular technology for the development of distributed, heterogeneous and always available systems. The application of agent technologies requires extensions to the existing object-oriented modeling languages to accommodate agent-related abstractions such as roles, organizations and environments. If it is difficult to analyze and establish the well-formedness of a set of diagrams of a UML-like object-oriented modeling language, it gets far more complex when the language is extended to add a set of agency related abstractions. This paper presents an ontology-based method for structuring MAS specifications. The goal of the method is to facilitate the analysis of such systems. The method proposes the analyses of MAS designs based on three phases that cover different sets of design properties. Focusing the analysis on related properties grouped into three different phases facilitates the design activity, the automatic detection of inconsistencies and the improvement of the design.

Keywords: Multi-agent systems, design, analysis, ontologies, modeling languages.

Resumo. O uso de agentes de software no desenvolvimento de sistemas distribuídos, heterogêneos e sempre disponíveis tem mostrado o quanto esta tecnologia pode ser útil. Porém sua aplicação requer a extensão das tecnologias orientadas a objetos, notadamente linguagens de modelagem, afim de descrever abstrações relacionadas a agentes, tais como papéis, organizações e ambientes. Considerando que análise e o estabelecimento da boa formação de diagramas descritos em linguagens de modelagens orientadas a objetos não é um problema simples, mais complexo ele se torna ao considerarmos tais linguagens acrescidas de um conjunto de abstrações de agência. Este artigo apresenta um método para estruturação de especificações de SMAs baseado em ontologias. O objetivo do método é facilitar a análise de modelos de design de SMAs, focando a análise em três fases distintas onde são considerados grupos de propriedades relacionadas. Desta forma, o método proporciona a detecção automática de inconsistências, melhorando a qualidade dos modelos de design e facilitando a atividade de modelagem.

Palavras-chave: Sistemas multi-agentes, design, análise, ontologias, linguagens de modelagem.

* This work has been sponsored by the Ministério de Ciência e Tecnologia da Presidência da República Federativa do Brasil, under 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

Object-orientation (OO) proved to be a powerful computational model for the development of real scale software systems. In order to help the design of such systems, several modeling languages for large OO applications have been consolidated. The UML standard [20] is an example that consists of a set of diagrams that capture different views of an OO software system. These views cover aspects such as how the system is to be used, how it is structured and how it will behave.

The models that represent real size OO applications using UML usually lead to very complex sets of diagrams whose well-formedness is very difficult to check, even independently of the applications. Until recently, the analysis of UML models to check the proper use of its many design artifacts and their allowed interrelationships has mostly been done in an ad hoc manner in successive versions of UML support tools such as [19]. Only recently, more systematic approaches to UML design checking have been developed [3][8][11][12][23]. They will be discussed in the related work section of this paper.

The nature of now-a-days distributed, heterogeneous, always available systems populated by autonomous components popularized the software agent and related abstractions (eg.: roles, organizations, environments) [21][24]. Since agents co-exist with objects for the solution of large scale distributed and heterogeneous systems, extensions of UML that incorporate the abstractions of the agent world have been proposed [2][22][24].

If the establishment of the well-formedness of a set of UML diagrams used to design a particular OO application is itself a difficult problem, it gets far more complex when UML is extended by adding the set of agency abstractions required by the new computational paradigm. The analysis of MAS designs represented by modeling languages that extend UML is indeed very complex and may compromise the adoption of the agent technology. Therefore, there is a need for an approach that facilitates the analysis of such designs by helping the designers to automatically detect and correct inconsistencies.

The paper presents an ontology-based method for structuring specifications of MAS. The goal of the method is to generate ontology-based specifications that facilitate the analysis of MAS. The method proposes the creation of such specifications along three phases. Each phase concerns itself with different sets of design properties. Focusing the analysis on related properties grouped into three different phases facilitates the design activity, the automatic detection of inconsistencies and the improvement of the design.

The generated specifications are created based on the specification of the MAS domain and on the specification of the modeling language being used to design MAS. Each generated specification is composed by an ontology [4][9] that specifies a set of domain and modeling language properties and queries used to analyze the designs according to additional set of properties. Designs are ontology instances created according to the ontology specification and analyzed by using the queries.

The MAS ontology-based specifications generated in the first phase of the proposed method describe a minimum set of concerns and axioms that characterize the MAS domain and the diagrams defined by the modeling language. In addition, such specifications also describe a set of queries that are used to analyze the design according to MAS domain properties and modeling language intra-model properties that were not described in the ontology. MAS domain properties characterize the MAS domain by defining the MAS entities, their properties and relationships. The modeling language intra-model properties fully specify the characteristics of each modeling language diagram by stating, for instance, the classes and the relationships that can be modeled in each static (structural) diagram.

In phase two, the design is analyzed according to the modeling language inter-model properties. Inter-model properties identify the interdependencies between the diagrams. Such properties state, for example, that an entity modeled in a given diagram must also be modeled in another diagram. The ontology generated in this phase contemplates all the MAS domain properties together with all the modeling language intra-model properties while the queries are used to evaluate the design according to inter-model properties. In *phase three*, the design is analyzed according to the compliance with designers' guidelines rules. The generated ontologies define all the MAS domain and intra-model properties already stated in the second ontology together with the modeling language inter-model properties. Queries are used to describe the set of design properties that provide guidelines to the designers to improve their models.

This paper is organized as follows. Section 2 presents an overview of the proposed method. Sections 3 , 4 and 5 explain the first, second and third phases of the method, respectively. Section 6 describes some related work and, finally, in Section 7 we present our conclusions and future work.

2 The Design Checking Method Overview

The proposed design checking method provides support for the analysis of MAS designs represented by modeling languages. The method is based on the specification of MAS and on the specification of the modeling language being used to design the MAS. By analyzing the designs according to both specifications, it is possible to check both the MAS properties (independently of the modeling language) and the modeling language properties themselves.

The specification of MAS is described by the TAO metamodel [21]. Such metamodel relates agents and its associated abstractions with objects while defining the main MAS entities (agents, organizations, environments, objects, agent roles and object roles), their properties and relationships (specialization, association, aggregation, dependency, play, control and ownership).

The modeling language specification is also based on a metamodel that describes the properties and characteristics of the language. In order to illustrate our approach, the modeling language being used in this paper is MAS-ML (Multi-Agent System Modeling Language) [22]. MAS-ML extends UML by including the agent related abstractions identified in TAO. The MAS-ML metamodel describes the artifacts (or diagrams) that are used to express both the structural and the behavioral aspects of MAS. MAS-ML defines three structural diagrams (the UML extended class diagram, organization diagram and role diagram) and two dynamic diagrams (the UML extended sequence and activity diagrams)

Both specifications – MAS domain and modeling language specifications – are represented by using ontologies and related queries. The use of ontologies to formalize the MASs domain and the modeling language is justified through the direct translation from design models into ontology instances and, consequently, the generation of knowledge-bases (KBs) that can be manipulated by using the reasoning services (or queries answers) that are available for this kind of data.

In Figure 1 below and in the following paragraphs, we provide mode details about the three phases of our proposed method. In phase one, the design is analyzed according to the MAS domain properties and to the modeling language intra-model properties by using queries. The ontology used in this phase (general ontology) identifies the MAS domain entities, properties and relationships as well as the modeling language diagrams.

In phase two, the design is analyzed according to the modeling language inter-model properties by using queries. Such analysis begins when the analysis of the first phase has been completed and the design has been updated according to the inconsistencies detected during the first phase. The ontology of phase two is an extension of the general ontology and it describes all MAS domain and modeling language intra-model properties.

In phase three, the design is analyzed by using queries to provide guidelines to the designer to improve his/her models. Phase three only begins after the design has been updated according to the inter-model inconsistencies found in phase two. The ontology being used in this phase is an extension of the phase two ontology and it specifies the MAS domain properties and all modeling language (intra and inter-model) properties.

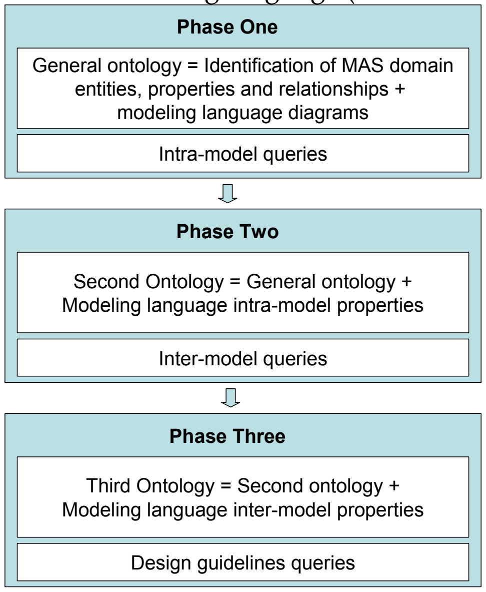
Phase One
General ontology = Identification of MAS domain entities, properties and relationships + modeling language diagrams
Intra-model queries
Phase Two
Second Ontology = General ontology + Modeling language intra-model properties
Inter-model queries
Phase Three
Third Ontology = Second ontology + Modeling language inter-model properties
Design guidelines queries

Figure 1. The ontology-based method

To formalize the specification of the MAS domain and the modeling language, we adopted Description Logics [1] because it is a decidable subset of first order logic and there is a recommendation from OMG [13] of using a DL-based language (OWL [15]) as a standard for ontologies description. Ontologies are represented by concepts, properties and axioms, where properties represent the relationships between the concepts and axioms represent the constraints over the concepts and relationships. Therefore, the ontologies that support the proposed method are described using a state-of-the-art DL reasoning system [17]. Considering C and D as ontology concepts and R as an ontology property, the meaning of the description language used in this paper and its associated syntax is shown in Table 1.

Table 1 – Description Language Syntax and Semantics

syntax	meaning
`(some R C)`	some individuals that are in the relationship R in the concept being described belong to concept C.
`(all R C)`	all of the individuals that are in the relationship R in the concept being described belong to concept C.
`(and C D)`	set of individuals obtained by intersecting the sets of individuals denoted by C and D
`(or C D)`	set of individuals which belong to the sets of individuals denoted by C or D
`(not C)`	set of individuals that does not belong to concept C
`(top )`	True
`(implies C D)`	individuals that belong to concept C also belong to concept D

3 Phase One: Analysing MAS Properties and Intra-model Properties

The ontology used in this phase partially formalizes the MAS domain by describing MAS entities' classes together with their instances and properties, and the relationships that can be used between them. The ontology does not fully formalize the MAS domain since it does not state the rules (or axioms) that are associated with the entities' properties and relationships. For instance, although the ontology identifies the relationships, it does not describe which entities can be linked by those relationships.

In addition, the ontology used in this phase briefly presents the modeling language being used by identifying the diagrams that it defines. The ontology does not completely specify the modeling language because it does not describe rules related to intra and inter-model properties.

The design produced by using the described ontology is, therefore, a MAS design, since it uses MAS abstractions defined in the ontology. However, such design may be not consistent with MAS domain properties and with the modeling language properties since the ontology does not describe axioms that guarantee such consistence. For that reason, queries are described and associated with the ontology to analyze the design and detect any MAS domain or intra-model inconsistence. The detection is automatically provided by the reasoning services from the DL-based system (RACER) and the query answer informs the designer where the inconsistencies are.

3.1 The General Ontology

As stated before, the ontology partially specifies the MAS domain and the modeling language. The ontology identifies, for instance, the entities' classes, the entities' instances and the relationships that link those entities. Figure 2 and Figure 3 illustrates two parts of the ontology that correspond to the specification of MAS domain properties. Figure 2 depicts the definition of agents, agent roles and organizations classes and instances while Figure 3 shows the play relationship identification. The classes, instances and relationships definitions are based on TAO.

The ontology also describes some modeling language properties by identifying the diagrams proposed in the modeling language. Figure 4 illustrates the three static diagrams defined in MAS-ML by stating that classes and relationships can be modeled in such diagrams.

```
(signature
  :atomic-concepts
    (class agent-class organization-class
       object-class environment-class role-class
       class-instance agent organization object
       environment agent-role object-role
...)...)...
(implies agent-role-class class)
(implies agent-class class)
(implies organization-class class)
(implies agent-role class-instance)
(implies agent class-instance)
(implies organization class-instance)
```

Figure 2. Identification of entities' classes and instances

```
(signature
   :atomic-concepts (...
     relationship inhabit play ownership
     specialization association aggregation control
     dependency
...) ...) ...
(implies play relationship)
```

Figure 3. Identification of the play relationship

```
(signature

   :atomic-concepts (...
  static-model class-model organization-model role-model)
   :roles (
(has-class :domain static-model
                    :range class
                    :inverse is-in-static-model)
        (has-relationship :domain static-model
                    :range relationship
                :inverse is-relationship-of)
...) ...) ...
(implies class-model static-model)
(implies organization-model static-model)
(implies role-model static-model)
```

Figure 4. Identification of MAS-ML static diagrams

3.2 The Intra-Model Queries

While describing the ontology in the previous section, the relationships were not completely specified. The ontology only states the available relationships but does not describe the entities that can be linked by them. It is an example of a MAS domain property that can be analyzed by using queries. Figure 5 illustrates query I that detects the play relationships that are not correctly used. The play relationship relates agent, sub-organization or objects to agent roles or object roles. By using the query it is possible to analyze the design and detects the play relationships that are not being used according to the specification.

Queries are also used in this phase to analyze the design according to the modeling language intra-model properties. An example of an intra-model property that is analyzed by queries is the definition of the classes and relationships that can be modeled in each static diagram. Figure 6 depicts query II that analyzes the design to detect if there is an organization diagram that has any association, dependency, aggregation or specialization relationship. Such relationships can not be modeled in organization diagrams. Such specification is described in the MAS-ML metamodel.

```
(retrieve (?play)
 (and (?play play)
      (?end1 ?play is-end1)
      (?end2 ?play is-end2)
      (or (?end1 (or role-class environment-class
             main-organization-class))
         (?end2 (or citizen-class environment-class))
           (and (?end1 agent-class)
                (?end2 object-role-class))
           (and (?end1 object-class)
                (?end2 agent-role-class))
)))
```

Figure 5. (Query I) Play relationship

```
(retrieve (?orgmd ?relation)
   (and (?orgmd organization-model)
        (?relation relationship)
        (?orgmd ?relation has-relationship)
        (?relation
    (or control association dependency
                 aggregation specialization))))
```

Figure 6. (Query II) Organization diagram

4 Phase Two: Analysing Inter-model Properties

Phase two should begin only after all inconsistencies detected at phase one have been solved. In this context, the ontology instance that represents the design has no MAS domain or intra-model inconsistencies. However, it is possible that some inter-model inconsistencies remain since inter-model properties have not been analyzed yet.

In order to guarantee that the ontology instance being used in this phase has not MAS domain or intra-model inconsistencies, the rules described by the queries used in phase one were transformed into axioms of the ontology used in phase two. The ontology used in this phase fully formalizes the MAS domain and the intra-model properties of the modeling language. Queries are used in this phase to analyze the design according to the inter-model properties that have not been checked yet.

4.1 The Second Ontology

The ontology described in this phase is an extension of the general ontology through the addition of new axioms defined based on the queries of phase one. In order to exemplify the second ontology, consider *queries I* and *II* described in Section 3.2 . These queries were transformed into axioms to formalize the play relationship (Figure 7) and the organization diagram (Figure 8). Figure 7 shows the ontology part that specifies that the play relationship can be used to link agents to agent roles, or sub-organizations to agent roles, or objects to object roles. Figure 8 illustrates the ontology part that describes that the organization diagram can be used while modeling any class and the play, ownership and inhabit relationships.

```
(signature :...
   :roles (has-end :domain relationship
                   :range class
          :inverse is-end)
          (has-end1 :parent has-end)
          (has-end2 :parent has-end)
...) ...
(implies play
  (or (and (all has-end1 agent-class)
           (all has-end2 agent-role-class))
      (and (all has-end1 sub-organization-class)
           (all has-end2 agent-role-class))
      (and (all has-end1 object-class)
           (all has-end2 object-role-class))
))
```

Figure 7. Formalization of play relationship

```
(implies organization-model (all has-class class))
(implies organization-model
                     (some has-relationship ownership))
(implies organization-model
                     (some has-relationship play))
(implies organization-model
                     (some has-relationship inhabit))
```

Figure 8. Formalization of organization diagram

4.2 The Inter-Model Queries

Since the MAS domain and intra-model properties are already described in the ontology axioms, it is now necessary to analyze the design according to the modeling language inter-model properties. Such properties state restrictions between the modeling diagrams. Figure 9 illustrates one of the inter-model properties that relate two static diagrams – the role and the organization diagrams. *Query III* analyzes the design to find out if there is an agent role class defined in a role diagram and not defined in an organization diagram. MAS-ML metamodel states that every role must be defined in an organization diagrams.

```
(retrieve (?agrl ?rlmd)
          (and (?agrl agent-role-class)
               (?rlmd role-model)
            (?rlmd ?agrl has-class)
            (not (?agrl
                       (some is-in-static-model
                             organization-model)))))
```

Figure 9. (Query III) Interdependence between
role and organization diagrams

Another example of an inter-model property is described in *query IV* (Figure 10) and relates a static diagram to a sequence diagram. *Query IV* analyzes the design to realize if there is an instance in a sequence diagram that is instance of a class that does not appear in at least one static diagram. MAS-ML metamodel defines that any instance must be an instance of a class modeled in one of the three static diagrams.

```
(retrieve (?ipath ?seqmd ?class)
          (and (?ipath instance-path
               (?clpath class-path)
               (?ipath ?clpath is-instanceOf)
               (?seqmd sequence-model)
               (?seqmd ?clpath has-path)
               (?clpath ?class has-head)
               (not (?class
               (some is-in-static-model static-model)))
))
```

Figure 10. (Query IV) Interdependence between
static and sequence diagrams

5 Phase Three: Analysing Well-formed Rules

The ontology instance (or design) being analyzed in phase three must be consistency with any MAS domain and modeling language properties. Phase three should start only after all inconsistencies detected at phase two have been solved, i.e., only after inter-model inconsistencies have been solved.

Similar to phase two, in order to guarantee that the ontology instance being used in this phase has no remaining inter-model inconsistencies, the rules described by the queries used in phase two were transformed into axioms of the ontology used in phase three. Therefore, the ontology instance analyzed in phase three obeys those axioms to be consistent with the ontology. The queries are used in this phase to analyze the ontology instance in order to suggest the designer how he/she could enhance it.

5.1 The Third Ontology

The ontology described in this phase is an extension of the second ontology through the addition of new axioms defined based on the queries described in the second phase. Such queries were transformed into axioms to formalize the interrelationships between static diagrams and between static and sequence diagrams. Figure 11 illustrates the formalization of the interdependence between role and organization diagrams described as query in *Query III* and Figure 12 depicts the formalization of the interdependence between static and sequence diagrams described as query in *Query IV.*

```
(implies (and agent-role-class

   (some is-class role-model))
(some is-class organization-model))
```

Figure 11. Formalization of interdependence between role and organization diagrams

```
(implies (and class-instance
           (some is-head
             (and path
               (some is-path sequence-model))))
         (some is-instanceOf
            (and class
               (all is-class static-model))))
```

Figure 12. Formalization of interdependence between static and sequence diagrams

5.2 The Design Guidelines Queries

The set of design rules will be formally encoded in queries in order to analyze the design and provide good design practices. Since the designs of multi-agent applications tend to be very complex, the design analysis can also be used to detect ill-structured design representations that can be replaced by structures recognized as good designed practices described in the set of rules. In this paper we present two examples of such queries.

Queries V (Figure 13) and *VI* (Figure 14) are examples of guidelines queries. *Query V* finds out the agent role class that was modeled in an organization diagram but was not modeled in a role diagram. Organization diagrams identify the roles that can be played by agents while role diagrams model the relationships between such roles. The relationships between the roles indicate how agents may interact. If there is an agent role class that does not appear in a role diagram, the agent that will play this role will not be able to interact with any other agent.

```
 (retrieve (?agrl ?orgmd)
   (and (?agrl agent-role-class)
        (?orgmd organization-model)
        (?orgmd ?agrl has-class)
        (not (?agrl
                (some is-in-static-model role-model)))))
```

Figure 13. (Query V) Agent roles in organization and role diagrams

Since an important characteristic of agents is the interaction with other agents while playing roles, the query helps the designer to discover the agent role classes that should be also modeled in role diagrams.

The objective of *Query VI* is to inform the designer about protocols that can be used to model agent interactions. In MAS-ML sequence diagrams, the interactions between agents are modeled based on protocols defined by the roles played by agents. Therefore, if the roles of the agents being modeled are identified, it is possible to suggest the protocols that the designer can use. Each agent role class defines a set of protocols that describes the sequence of messages sent to and received from another agent role.

```
(retrieve (?prtcl ?seqmd ?agpath)

      (and (?prtcl agent-protocol)
           (?seqmd sequence-model)
           (?agpath agent-instance-path)
           (?clpath agent-class-path)
           (?agpath ?clpath is-instanceOf)
           (?seqmd ?clpath has-path)
           (?clpath ?agrolepath has-tail)
           (?agrolepath ?agroleclass has-head)
           (?agroleclass ?prtcl has-protocol)))
```

Figure 14. (Query VI) Agent role protocols

6 RELATED WORK

Dong and colleagues [6] used Z and the theorem proving Z/EVES to verify domain ontologies coded in DAML+OIL [5]. They defined the Z semantics for the DAML+OIL language primitives and their associated constraints to check if the ontology definition is according to them. In this sense, they check the static (or structural) part of the ontology which means class (concept) inconsistency, subsumption and instantiation testing. They also show that it is possible to check other ontology properties by defining theorems that relate ontology classes and roles. Our work is related to theirs in the sense that we define the semantics for an MAS modeling language metamodel with its associated diagrams and constraints using a DL-based language to check if the design models coded in that modeling language and translated to DL are consistent with their metamodel. We use a DL reasoner and its associated reasoning services to perform the automatic checking of such models combined with a list of pre-defined queries. Our approach allows not only the checking of structural properties of the models, but some dynamic properties as well.

Kalfoglou and Robertson [10] used ontologies to reason about domain specifications correctness. They considered the correctness of an application specification relatively to the application domain. In this sense, they propose the use of an ontology that describes the application domain to guide the specification engineer. Therefore, as they are considering a formal specification for the application which is based on an ontology which describes the application domain, they can automatically check the existence of ontological inconsistencies in the application specification. Considering the four layer cake of the metadata architecture from OMG-MOF [14], their work navigates between the domain model layer (M1) and the instance layer (M0) while ours navigates between the metamodel layer (M2) and the domain model layer (M1), which means that we are considering the overall class of MAS applications, independently of the considered application domain.

Since modeling languages do not have a precise semantics yet, several works address the problem of design models verification [3][8][11][12][23]. Kim and Carrington [11] give a translation from a UML class model to an Object-Z specification, but they don't provide means to verify the model. Our work defines an ontology-based method that provides a formal description of MAS design models and uses knowledge-based reasoning techniques to verify these models consistency. Ekenberg and Johannesson [8] define 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. We define an ontology-based method that uses an ontology description language based on DL, which is a decidable subset of FOL. The translation of MAS design models to the ontology description language can be done automatically by using systems such as RICE [18] or Protégé [16] with RACER, among others. Also, the verification of consistency is automated by applying the reasoning and inference services the generated KB.

Mens, Straeten and Simmonds [12][23] use 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 [23] 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. Their work related to ours in the way they check consistency between models. Our work considers an MAS context and extends the idea of *classless instance* when, for example, we verify the absence, in any organization diagram, of classes that were predefined in role diagrams or class diagrams. Berardi [3] uses DL to formally describe a UML class diagram and the CORBA-FaCT [7] and the RACER system to reason about them in order to classify the models concerning their consistency. Such approach is very similar to ours since the diagram description using DL could be considered an ontology for the UML class diagrams. However, while they provide support for verification of a class of models according to an object-oriented metamodel we do the same for all possible models according to a multi-agent-oriented metamodel.

7 CONCLUSIONS AND FUTURE WORK

This paper presents a three phase ontology-based method for structuring formal specifications of MAS based on ontologies that describe the MAS domain and the metamodel of a MAS UML-like modeling language. The proposed method is composed of three phases that support the desired flexibility during the design activity. Such flexibility allows syntactical incorrectness during the creation of design models. The models themselves are checked in phase 1 (by analyzing the intra-model properties) while the interrelationships between the models are checked in phase 2 (by analyzing the inter-model properties). Finally, the method also gives support to the definition of design guidelines in phase three. Such guidelines are good practices of design using the modeling language.

We are developing a MAS-ML graphical tool where designers could model MAS using the MAS-ML diagrams [22]. The proposed method provides a back-end for the tool allowing the analysis of the models during their building. Thus, the inconsistencies that arise during design construction will be automatically detected, which will help designer not even to decrease the time of building but to improve quality of MAS designs, as well.

ACKNOWLEDGMENTS

This work is partially supported by CNPq/Brazil under the project "ESSMA", number 5520681/2002-0.

REFERENCES

[1] BAADER, F., CALVANESE, D. MCGUINESS, D., NARDI, D. and PATEL-SCHNEIDER, P. **The description Logic Handbook – Theory, Implementation and Applications**, Cambridge Univ. Press, 2003

[2] BAUER, B. MÜLLER, J.P. and ODELL, J. Agent UML: A Formalism for Specifying Multiagent Software Systems In: Ciancarini and Wooldridge (Eds) **Agent-Oriented Software Engineering**, Springer-Verlag, LNCS vol 1957, 2001.

[3] BERARDI,D. Using DLs to reason on UML class diagrams, in **Proceedings of th KI-2002 Workshop on Applications of Description Logics**, 2002.

[4] BORST, W.N. **Construction of Engineering Ontologies**, University of Twente, Enschede, NL, Center for Telematica and Information Technology, 1997.

[5] DAM+OIL – DARPA Agent Markup Language http://www.daml.org/

[6] DONG,J.S., LEE,C.H., LI, Y.F. and WANG, H. Verifying DAML+OIL and Beyond in Z/EVES, In **Proceedings of the 26th International Conference on Software Engineering** (ICSE'04), pp. 201-210, 2004.

[7] FaCT – Fast Classification of Terminologies, available at: http://www.cs.man.ac.uk/~horrocks/FaCT

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

[9] GRUBER,T.R. A Translation Approach to Portable Ontology Specification, **Knowledge Acquisition**, 5, 199-220, 1993.

[10] KALFOGLOU, Y. and ROBERTSON, D. A case study in applying ontologies to augment and reason about the correctness of specifications, **in Proceedings of the 11th International Conference on Software Engineering and Knowledge Engineering (SEKE99)**, Kaiserlautern, Germany, 1999, available at http://www.ecs.soton.ac.uk/people/yk1/ (04/19/2005).

[11] KIM,S and CARRINGTON,D. 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.

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

[13] OBJECT MANAGEMENT GROUP – OMG http://www.omg.org/

[14] OBJECT MANAGMENT GROUP: OMG – Meta Object Facility (MOF) Specification, version 1.4, available at http://www.omg.org/cgi-bin/doc?formal/2002-04-03 (last visited 04/25/2005).

[15] OWL –Ontology Web Language, available at http://www.w3c.org/TR/owl-features/

[16] PROTÉGÉ 2000 Ontology Editor – available at http://protege.stanford.edu/

[17] RACER, Renamed Abox and Concept Expression Reasoner, available at: http://www.sts.tu-harburg.de/~r.f.moeller/racer

[18] RICE, Racer Interactive Client Environment http://www.cs.concordia.ca/~haarslev/racer/rice.jar

[19] RATIONAL Rose. http://www-306.ibm.com/software/rational/

[20] RUMBAUGH,J., Jacobson,I. and Booch,G. **The Unified Modeling Language Reference Manual**. Addison-Wesley, 1999.

[21] SILVA, V., GARCIA, A., BRANDÃO, A., CHAVEZ, C., LUCENA, C., ALENCAR, P. Taming Agents and Objects in Software, in Garcia, Lucena, et al (Eds) **Software Engineering for Large-Scale Multi-agent Systems- Research Issues and Practical Applications**, Lecture Notes in Computer Science, vol 2603, 2003.

[22] SILVA, V. and LUCENA, C. From a Conceptual Framework for Agents and Objects to a Multi-Agent System Modeling Language, In: Sycara, K., Wooldridge, M. (Eds.), **Journal of Autonomous Agents and Multi-Agent Systems**, 9, 145-189, 2004.

[23] STRAETEN,R. and SIMMONDS,J. Detecting Inconsistencies between UML Models Using Description Logic, in THE 2003 INTERNATIONAL WORKSHOP ON DESCRIPTION LOGICS, available at http://CEUR-WS.org, last access 07/15/05

[24] WAGNER, G. The Agent-Object-Relationship Metamodel: Towards a Unified View of State and Behavior, **Information Systems**, Vol 28, 5, 475 – 504, 2003.

[25] ZAMBONELLI, F., JENNINGS, N. and WOOLDRIDGE, M. Organizational Abstractions for the Analysis and Design of Multi-Agent Systems, In: Ciancarini and Wooldridge (Eds) **Agent-Oriented Software Engineering**, Springer-Verlag, LNCS vol 1957, 2001.