

Raising the level of abstraction: On the application of UML for multiagent systems design

JJ Sampson

Norwegian University of Science and Technology (NTNU)

Trondheim, Norway

sampsonj@idi.ntnu.no

Abstract

Current research in the field of multiagent systems design has a predisposition towards borrowing methods, frameworks, notation, terminology, profiles, models and languages from the object oriented discipline (e.g. Baclawski *et al.*, 2002; Cranefield, 1999; Sardinha *et al.*, 2003; Wooldridge 2002). This paper will describe the application of UML in the area of multiagent systems (MAS). The research question to be addressed is: how is UML, an object oriented modelling language, useful in the context of multiagent systems design? In future research we endeavour to measure the appropriateness of UML, primarily through validation and verification techniques such as external correctness - “is it the right model” and internal correctness “is the model right”. This research is important because development tools and methods play an important part in agent-based systems, indeed choosing and applying the right techniques is non trivial.

1. Introduction

A multiagent system comprises agents which interact in an environment through different “spheres of influence” where different agents can influence or control a part of an environment (Jennings, 2000). Because these spheres of influence may coincide and bring about dependency relationships among other agents it is inherently important to understand the type of interaction that takes place between agents (Wooldridge, 2002). Due to the novelty of multiagent systems we cannot reasonably conclude that existing approaches and techniques in more traditional analysis and design are sufficient.

The research question to be addressed in this paper is: how is UML, an object oriented modelling language, useful in the context of multiagent systems design. The importance of this research is recognised by the Object Management Group (OMG), who claims we need to determine the kinds of modelling languages required as well as meta-models necessary for supporting agent specification and implementation.

2. The Unified Modelling Language (UML)

The Unified Modelling Language is a standard language for writing software blueprints (Booch, Rumbaugh and Jacobson, 1999) whose constructs originate from object oriented programming. The OMG (2003) state that UML is a language for documenting: software

systems, modelling business systems and other non-software systems. The language is for specifying and constructing object-oriented systems, and also for visualising and documenting such systems. The three building blocks of the language are: things, relationships and diagrams. Booch, Rumbaugh and Jacobson (1999) describe the three building blocks, firstly, ‘things’ are the abstractions that are first-class citizens in a model; relationships link the ‘things’ together and the diagrams group ‘interesting’ collections of things. In UML ‘things’ may be either: structural¹, behavioural, grouping or annotational. The UML defines several graphical diagrams such as: the use case diagram, the class diagram, behaviour diagrams and implementation diagrams².

The motivation behind the development of the object paradigm was to remove from the user the need to deal with machine oriented constructs, and focus towards a more human oriented view, using objects, that more closely reflect the world. Indeed the main claim has been to raise the level of abstraction from which we model the world. While this has been successful in programming languages, we cannot be certain that the same techniques can be applied, as successfully, to multiagent systems.

3. The wood or the trees: agents versus objects

Are agents merely objects endowed with more (smarter) functionality? Many researchers question the properties an object must possess in order to be defined an agent. However as there is no accepted formal definition of the term agent, it is difficult to decide what properties can be attributed to objects. Nonetheless, Wooldridge and Jennings (1995) broadly define an agent as “a computer system that is *situated* in some *environment*, and that is capable of *autonomous action* in this environment in order to meet its design objectives”. Whereas an object may be defined as “computational entities that *encapsulate* some state, are able to perform actions, or *methods* on this state, and communicate by message passing” (Wooldridge, 2002).

Agents use agent communication languages (ACL’s) to communicate whereas objects use message passing. However, the Agent Working Group (AWG) has undertaken research to determine the plausibility of defining entities which can communicate using both ACL’s and message passing. In this way it may be possible to build a system with agents

¹ There are seven structural ‘things’ in UML: classes, interfaces, collaborations, use cases, active classes, components and nodes.

² Refer chapter two, pp.24-26 of Booch, Rumbaugh and Jacobson (1999) for detailed description.

communicating with other agents using ACL's and also with objects using message passing (OMG AWG, 2000). They claim that this would allow agents and objects to interoperate as first class objects.

There are several forms of agent technology important for systems development. However, for the purposes of this research we are interested in adaptive, distributed agents with beliefs, goals, plans and assumptions. By adaptive we mean the ability of a service to adapt to the users, service providers and communication context. Such "intelligent"³ agents require a basic set of attributes and facilities (OMG AWG, 2000).

The most significant difference between agents and objects is the degree to which they are autonomous (Wooldridge, 2002). Specifically an object does not have control over its behaviour whereas an agent does. While it is possible to construct agents using object-oriented techniques, (e.g. changing when a method executes by using decision making techniques) the basic premise of the object-oriented model is violated.

Another distinction between agents and objects is the notion of *flexible* autonomous behaviour (Wooldridge, 2002). At present the standard object model does not consider reactive, proactive and social behaviours within and between objects. However, these features can be forced on the model, but once again violating the basic premise of the object-oriented model. Likewise, Medvidovic *et al.*, succinctly comment,

"UML's genesis and its primary strength is modelling software from an object-oriented perspective, where the major system elements (components), their constituent building blocks (subcomponents), their interactions (connectors), and the data exchanged among them are all represented in the same way – as objects" (p.51).

Most of the current literature describes how object technology may be used to implement agent technology. However, we must note that although objects and agents are alike in several ways they also differ in certain important aspects. Agents, objects and entities can be perceived to be the same or similar conceptually as they have a name, an identity, a state and behaviour. Nonetheless, the OMG AWG⁴ claims that agents are different from objects from a conceptual modeling perspective and an implementation perspective. Some of the differences they suggest are:

³ The notion of intelligence suggests that these agents exhibit attributes such as beliefs, desires and intentions which affect how the agent acts in a particular situation.

⁴ AWG - Agent Working Group, a sub-group of the Object Management Group.

- Agents are considered to be more functional than single objects, exhibiting more ‘intelligent’ behaviour.
- Agents are autonomous, they can decide whether to respond to messages from other agents, whereas objects generally perform when they are asked to. Agents are also reactive, which means they are always ‘listening’ and participating, containing their own threading capability.
- Intelligent agents use an ACL (agent communication language) to communicate with other agents. This language is very powerful and supports the complex belief-desire-intention information, allowing the agent to use inference and also exhibit learned behaviour. While objects can evolve and take on new roles this functionality is not well supported in many object systems, objects use a fixed set of messages to communicate.
- In many agent systems, agents are represented at the object instance or individual level, this means there is no strongly typed class concept. Agents tend to become specialised through their content and behaviour rather than being defined as specific ‘staff agents’ or ‘resource agents’.
- Inheritance is not commonly viewed as an intrinsic agent aspect, but agents can be manipulated to support some types of inheritance.

(Adapted from: OMG AWG, 2000)

The OMG AWG argue that whether an agent is implemented directly, or as an object with intelligent interpreters, there is still a fundamental difference, such that agents are a “smart class of objects” that communicate using specific agent communications. While agents and agent systems may be composed of objects and object infrastructure, they are fundamentally different from objects in that they can be described as existing at a higher level of abstraction. While we acknowledge the debate on the differences between agents and objects, the need for interoperability and a standard common ontology are of more importance to our research in multiagent systems. Indeed, due to widespread industry use of object technology it makes sense to build agent systems using object technology and to make such systems compatible with the infrastructure used for complex object-oriented software. But we recognise that further research is necessary to determine the strength of the object-oriented paradigm for supporting multiagent systems and to determine whether other tools and infrastructures may be more suitable (refer section 6).

4. Modelling agents

Currently an agent-based unified modelling language (AUML) is being developed by the FIPA Modelling technical committee (www.auml.org) in response to a need from system designers for a tool which can more adequately capture the idiosyncrasies of multiagent

systems. However little effort into investigating the use or otherwise of alternative tools and techniques for multiagent systems development has been researched.

In addition researchers have been looking to extend UML not only for agent-based systems, but to standardise the language as a design language for software architecture modelling (e.g. Medvidovic *et al.*, 2002).

4.1 Extending UML for ontology modelling

Ontology can be described as ‘what there is’, and relies on the use of specific terms to construct a description of reality. Most research in AI tends to define the notion of ontology through philosophy in particular that of Bunge⁵ (1977). I claim that the use of such a philosophical ontology as a theoretical basis for multiagent systems is inappropriate because we are not representing reality when developing multiagent systems, but the way agents reason about knowledge, interact as a ‘society’ and make decisions. In future research we will determine the efficacy of determining a psychological ontology (Veres and Hitchman, 2002) as the theoretical basis for multiagent systems and the semantic web.

Nonetheless, defining ontologies, and choosing the most appropriate language to express the ontology is one of the most important current research areas in multiagent systems. This is because in order for an agent to comprehend a message received, the nomenclature of the message must make sense to the agent. For example, to ensure that one agent’s understanding of the concept ‘snow’ is the same as another agent’s understanding of the concept requires a standard shared ontology. The importance of such a shared communication ontology is to ensure that the interpretation brings about the same change (or not), in structure and or behaviour, in both the sender and the receiver (Pastor, J.A., Taylor, S.L., McKay, D.P., McEntire, R., 1997).

Research has been undertaken using UML as the language of choice to model ontologies within the field of software agents. However differences in the design goals and origins of UML compared with ontology languages (e.g. IEEE SUO, DAML, SUMO) often complicate the mapping. Alternative approaches for ontology modelling have traditionally been the Knowledge Interchange Format (KIF) and description logic. Ontologies for agent communication typically describe a taxonomy of class and subclass relations together with the definitions of the relationships between the classes. As mentioned previously on of the

⁵ Bunge, M. (1977): Treatise on basic philosophy. III: Ontology: The furniture of the world. Reidel, Dordrecht.

reasons for developing ontologies in this context is so agents can share knowledge with other agents.

Cranefield and Purvis (1999) claim that because UML is now widespread in industry and provides “an effective and scalable approach to conceptual modelling” it “therefore warrants serious consideration as an ontology modelling language”. However, UML is highly solution based, and is not as intuitive as other modelling languages.

While UML was originally intended as a model communication tool for users during the development of systems in object-oriented programming languages, current research has been invested in removing one of the most commonly stated criticisms of the suitability of UML for representing formal models such as ontologies. The OMG Model Driven Architecture (MDA)⁶ has been striving to define UML models in a more formal, machine-processable form which can be used at compile and runtime (Cranefield 1999). In addition, it is claimed that UML can support the generation of application and middleware code and translation of data in a heterogeneous environment⁷.

DAML⁸+OIL⁹ is a formal ontology designed for web-enabled agents who can reason about knowledge and dynamically integrate services at run-time (Baclawski, *et al.*, 2002). Current knowledge representation ontologies (e.g. RDF, DAML+OIL) do not have any graphical representation, thus the incentive to utilise UML’s popularity and graphical notation. For example, Baclawski, *et al.*, (2002) have been undertaking research to use UML class diagrams to develop and display complex DAML+OIL ontologies. They have also researched the differences between UML and DAML+OIL (refer Table 1) and provide initial suggestions on how to overcome the differences.

While there are similarities between UML and knowledge representation languages, the fundamental problem with extending UML for ontology development, is due to the important distinction between knowledge representation approaches and object-oriented approaches: that of monotocity (Baclawski, *et al.*, 2002). They note, “[a] logical system is monotonic if adding new facts can never cause previous facts to be falsified” (*ibid.*, p.8).

“Both RDF and DAML+OIL are monotonic: asserting a new fact can never cause a previously known fact to become false. By contrast, UML and other OO systems

6 <http://www.omg.org/mda/>

7 <http://www.omg.org/technology/cwm/>

8 DARPA Agent Markup Language (DAML)

9 McGuinness D. L., Fikes, R., Hendler, J., and Stein, L.A. (2002): DAML+OIL: An ontology language for the semantic web. IEEE Intelligent Systems, September/October: 72–80. (cited in Baclawski,*et al.*)

are typically not monotonic. There are many forms of nonmonotonic logic, but the one that is closest to UML and OO systems is a logic that assumes a closed world” (Baclawski, *et al.*, 2002, p.8)¹⁰

While Baclawski, *et al.*, (2002), attempted to provide rules for translating UML concepts to DAML+OIL concepts they came to the conclusion that some of the concepts are significantly incompatible. For example, the ‘property’ concept of DAML+OIL although similar to the UML ‘association’ concept, they claim it cannot be mapped easily. Indeed, they suggest, “this is the main obstacle to using UML (and UML tools) for DAML-based ontology development” (p.13). However they do propose extensions (refer appendix 1, Figure 2 and Figure 3) to the UML metamodel, with the motivation of making UML more acceptable to the knowledge representation community as the “preferred graphical notation for KR languages” (*ibid.*, p.13).

DAML Concept	Similar UML Concepts
Ontology	Package
Class	Class
As Sets (union, intersection, etc.)	Not Supported
Hierarchy	Class Generalization Relations
Property	Aspects of Attributes, Associations and Classes
Hierarchy	None for Attributes, limited Generalization for Associations, Class Generalization Relations
Restriction	Constrains Association Ends, including multiplicity and roles. Implicitly as class containing the attribute
Data Types	Data Types
Instances and Values	Object Instances and Attribute Values

Table 1: High-level Mapping of UML and DAML concepts (Baclawski, *et al.*, 2002, p.2)

The recommendation by Baclawski *et al.*, (2002) for a modification to the UML metamodel is to enable a designer to model using first-class properties as well as to construct classifiers using boolean operations and quantification.

4.2 Communication and application ontologies

Another important distinction often confused in the literature is the difference between shared communication ontologies and application ontologies. Pastor, *et al.*, (1997) defined an

¹⁰ Baclawski, *et al.* (2002), use a simple example to explain how monotonic logic considers requirements such as the possibility of not having a name (refer page 8).

intelligent resource agent architecture that specifies a common ontology in a representation that subsumes the models of the individual agents. In their application they have begun to introduce a separation between the *application* ontology manipulated by the agents themselves and the *communication* ontology used for inter-agent communication. They view,

“the ability to translate bi-directionally between the communication ontology and a collection of application ontologies as being critical to the long-term success of distributed agent architectures, since it places few—if any—restrictions on the kinds of applications that can participate as agents: as long as each agent supports mappings between its own ontology and the communication ontology, it can communicate with any other agent in terms of the communication ontology” (Pastor, *et al.*, 1997).

The C++ class architecture for the Intelligent Resource Agent system (Pastor *et al.*, 1997) is shown at Figure 1. In an object’s definition they have included both the methods to be invoked on that object and the communication act necessary to satisfy the successful execution of each method. For example, a domain-relevant class such as ‘Resource’ which implements a resource object in a resource catalog, can be defined in terms of its unique defining attributes, and inherit its communication and database aspects (*ibid.*). Figure 1 shows both the structure of the knowledge-base objects used to depict the abstract ontology, and the structure of the application-level objects in the object-oriented language C++. In this architecture agents influence objects that are isomorphic, in terms of structure and taxonomy, to the underlying domain ontology (Pastor *et al.*, 1997).

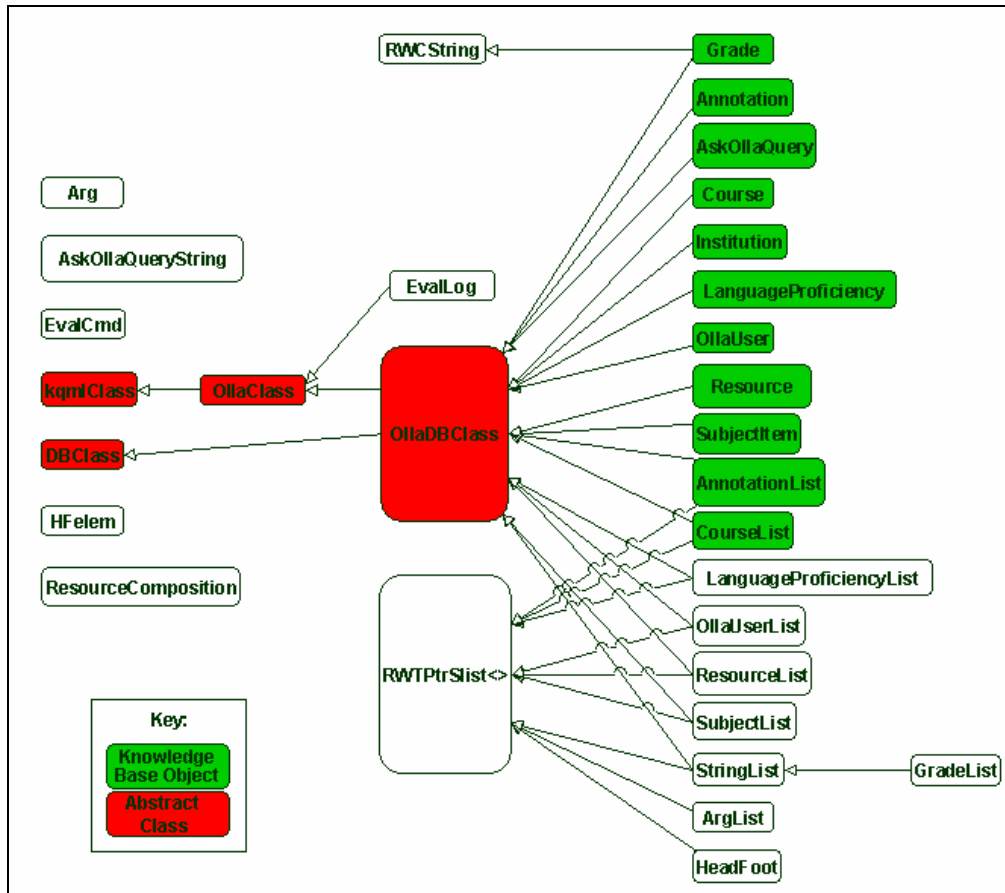


Figure 1: Shared Domain Ontology (Pastor, *et al.*, 20** , p.)

5. Frameworks and Methodologies for multiagent systems

There is currently no agreement on the kinds of concepts that an agent-oriented methodology should support. Wooldridge (2002) comments that “the kinds of concepts and notations supported by UML are not necessarily those best suited to the development of agent systems” (p.242).

In 1999 Wooldridge *et al.*, claimed that if agents were to realise their potential as a software engineering paradigm, then specific techniques were required to develop them. More precisely they claimed that existing software development techniques such as object-oriented analysis and design were unsuitable for agent based development. They comment “[t]here is a fundamental mismatch between the concepts used by object-oriented developers ...and the agent-oriented view” (*ibid.*, p.1). The problem with existing approaches is that they fail to address an “agent’s flexible, autonomous problem-solving behaviour, the richness of the interactions, and the complexity of agents system’s organizational structures”. The Gaia methodology was hence defined to address these four complex issues.

The Gaia methodology (Wooldridge, Jennings and Kinny, 1999) utilises some terminology and notation from object-oriented analysis and design, however “it is not simply a naive attempt to apply such methods to agent-oriented development. Rather, it provides an agent-specific set of concepts through which a software engineer can understand and model a complex system” (Wooldridge, 2002, p.228). The methodology is for building models in the analysis and design phase with an emphasis towards the development of agent-based systems as a “process of *organizational design*” (*ibid.*, p.229). Wooldridge (2002) suggests that the main Gaian concepts can be categorised as either abstract or concrete (shown at Table 2). The abstract concepts are associated with the analysis phase of the methodology and the concrete concepts are associated with the design and implementation.

Abstract concepts	Concrete concepts
Roles	Agent types
Permissions	Services
Responsibilities	Acquaintances
Protocols	
Activities	
Liveness properties	
Safety Properties	

Table 2: Abstract and Concrete Concepts in Gaia (Wooldridge, 2002, p. 229).

The design stage of the Gaia methodology involves three main steps, firstly the creation of an agent model¹¹ which includes aggregating the previously defined roles into agent types forming an agent type hierarchy. Wooldridge *et al.*, (1999) comment that the agent model is to record the various agent types that will be used in the system and the agent instances that will realise these agent types at run-time. The second and third steps in the design stage include the development of a services model and an acquaintance model¹² (refer Wooldridge *et al.*, (1999) for a detailed description). Several other approaches have been defined for multiagent systems design, however, these approaches like the Gaia methodology “do not attempt to unify the analysis and design of a multiagent system with its design and implementation within a particular agent technology” (*ibid.*). However, due to the proliferation of agent technologies Wooldridge *et al.*, (1999) advocate the use of such general methodologies.

Sardinha *et al.*, (2003) use the object-oriented framework of Fayad (1999) for implementing a communication infrastructure for agents. They claim that the framework allows an “easy

¹¹ refer appendix 2, Figure 4, for an example agent model

¹² refer appendix 2, Figure 5, for an example acquaintance model.

mapping of models” (*ibid.* p.86) developed by the analysis and design phase of the Gaia Methodology to object-oriented code. However, from their work it is unclear what they mean by neither the notion of ‘framework’, nor how to implement ‘easily’ the models defined by the Gaia methodology.

6. Conclusion and further research

The research question addressed in this paper was: how is UML, an object oriented modelling language, useful in the context of multiagent systems design. This paper has described some of the current applications for UML in the context of multiagent systems design; such as an ontology development tool, techniques within a methodology (e.g. Gaia) and as part of an overall framework for development. In addition this paper discussed the differences between objects and agents, where the traditional view of an object was compared with that of Wooldridge (2002) and the OMG AWG’s. They claim that objects and agents are distinct in at least three ways. Firstly, agents are more autonomous than objects, implying that agents decide for themselves whether or not to perform an action on request from another agent. Secondly, agents are capable of flexible behaviour which suggests they are reactive, proactive and social. Thirdly a multiagent system is essentially multi-threaded where each agent is assumed to have at least one thread of control (*ibid.* p.27). While UML is popular amongst developers in industry some communities such as the knowledge representation community have not adopted UML as readily because UML currently lacks a formal definition. Nonetheless research is underway to define an agent based UML (e.g. the agent working group www.auml.org).

Despite its popularity as a graphical modelling language, research is required to determine the efficacy of UML for multiagent systems design. Baclawski *et al.*, (2002) are working at extending UML as an ontology development tool, however there are some fundamental issues between what UML was originally intended to be used for and that which researchers and practitioners are attempting to achieve with the language.

The next phase of this research will be to examine the appropriateness of using alternative modelling languages and tools for developing an ontology for agent communication and application in the context of the ADIS project <http://adis.idi.ntnu.no/>. Furthermore, we are interested in determining the use or otherwise of a psychological based ontology for multiagent systems. The Referent Modelling Language (RML) (Sølvberg, 1993) is a powerful,

graphical, conceptual modelling tool which we endeavour to use as a language for representing an agent ontology suitable for the ADIS project. The appeal of using RML for this purpose is the graphical notation which can be directly mapped into logic statements which can then be implemented directly using Prolog.

7. References

- Baclawski, K., Kokar, M., and Smith, J., (2001): "*Metamodeling Facilities*" working paper presented at OMG Technical Meeting, <http://vis.home.mindspring.com/>
- Baclawski, K., Kokar, M.K., Kogut, P.A., Hart, L., Smith, J., and Letkowski, J. (2002): Extending the Unified Modelling Language for ontology development, *Software Systems Model, Special Issue UML 2002*, vol 1, pp. 1–15.
- Cranefield, S. and Purvis, M. (1999): UML as an ontology modelling language. In Proceedings of the Workshop on Intelligent Information Integration, 16th International Joint Conference on Artificial Intelligence (IJCAI-99).
- Fayad, M. and Schmidt, D. (1999): *Building Application Frameworks: Object-Oriented Foundations of Design*. First Edition, John Wiley & Sons, (cited in Sardinha *et al.*).
- Jennings (2000): On agent-based software engineering. *Artificial Intelligence*, vol 117, pp.277-296.
- Medvidovic, N., Rosenblum, D.S., Redmiles, D.F. and Robbins, J.E. (2002): Modeling Software Architectures in the Unified Modeling Language, *ACM Transactions on Software Engineering and Methodology*, vol, 11. 1, pp. 2-57.
- Odell, J. H., Van Dyke P., and Bauer, B. (2001): Representing Agent Interaction Protocols in UML, *Agent-Oriented Software Engineering*, Paolo Ciancarini and Michael Wooldridge ed., Springer, Berlin, pp. 121-140.
- Odell, J., Parunak, H.V.D., Bauer, B. (2000): "Extending UML for Agents," Proc. of the Agent-Oriented Information Systems Workshop at the 17th National conference on Artificial Intelligence, Gerd Wagner, Yves Lesperance, and Eric Yu eds., Austin, pp. 3-17.
- OMG (2003): *OMG-Unified Modeling Language Specification*, v1.5. An Adopted Formal Specification of the Object Management Group, Inc.
- OMG AWG (2000): *Agent Technology* – The Agent Working Group, Object Management Group (OMG) Green paper, version 1, available at http://www.objs.com/agent/agents_Green_Paper_v100.doc
- Parunak H.V.D. and Odell, J. (2002): "Representing Social Structures in UML," Agent-Oriented Software Engineering Workshop II, Michael Wooldridge, Paolo Ciancarini, and Gerhard Weiss, eds., Springer, Berlin, 2002, pp. 1-16.
- Pastor, J.A., Taylor, S.L., McKay, D.P., and McEntire, R. (1997): An Architecture for Intelligent Resource Agents, *2nd IFCS International Conference on Cooperative Information Systems (CoopIS '97)*
- Sardinha, J.A.R.P., Ribeiro, P.C., Milidiu, R.L. and Lucena, C.J.P.L. (2003): An Object-Oriented Framework for Building Software Agents, *Journal of Object Technology*, vol. 2, no.1, pp.85-97.
- Sølvberg (1993): An Introduction to Concept Modelling for Information Systems, *Information Systems Engineering*, Springer Verlag, Berlin-Heidelberg.
- Veres, C., and Hitchman, S. (2002): A psychological approach to conceptual modelling, In: *Proceeding of the European Conference on Information Systems (ECIS 2002)*, Gdansk, Poland.
- Wooldridge, M., Jennings, N.R. and Kinny, D. (1999): A methodology for agent-oriented analysis and design. In *Proceedings of the 3rd International Conference on Autonomous Agents (Agents 99)*, Seattle, WA, pp. 69-76.
- Wooldridge, M. (2002): *An Introduction to Multiagent Systems*, John Wiley & Sons, England.

Appendix 1: The UML metamodel extensions recommended by Baclawski *et al.*, 2002.

The extensions are: Property, Restriction, Union, Intersection and Complement.

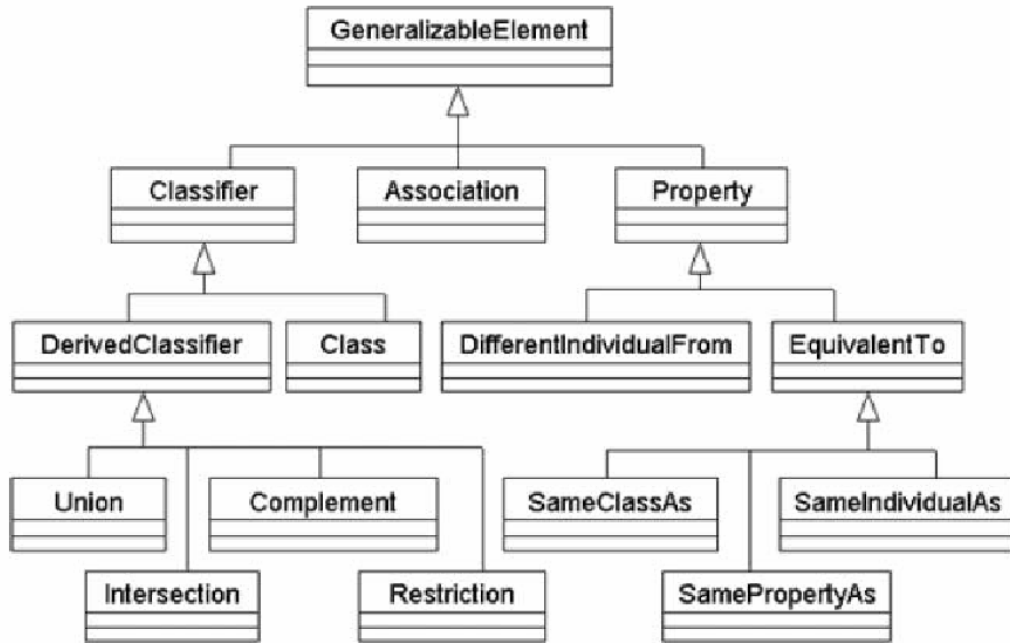


Figure 2: The specification of the proposed UML Modification: Metaclass Hierarchy (Baclawski *et al.*, 2002, p.11).

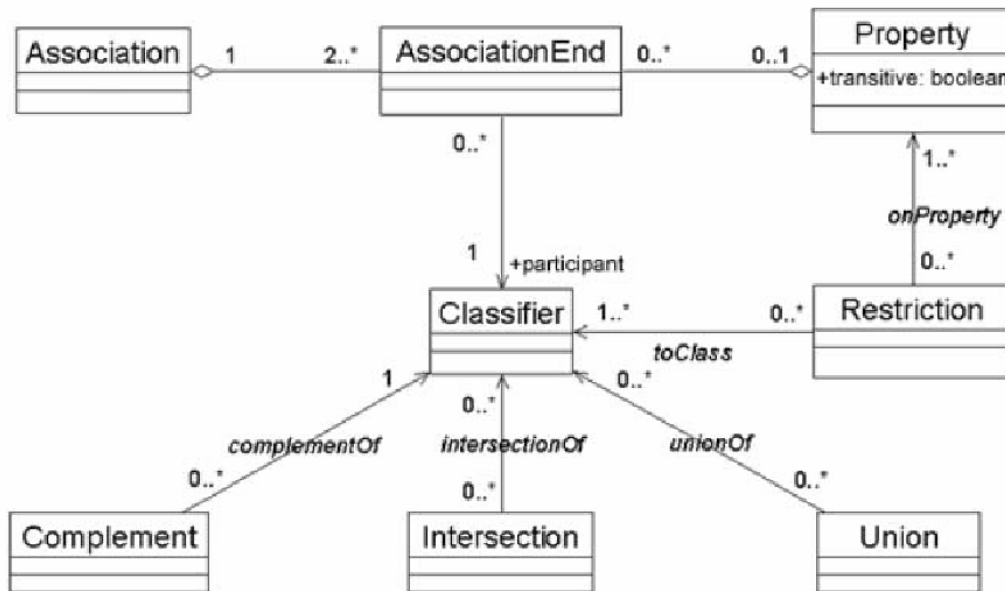


Figure 3: The specification of the proposed UML Modification: Meta-Associations (Baclawski *et al.*, 2002, p.11)

Appendix 2: Examples of the design models developed by Wooldridge *et al.* (1999)

Wooldridge *et al.*, (1999) used the Gaia methodology for the analysis and design of an agent-based system for managing a British Telecom business process.

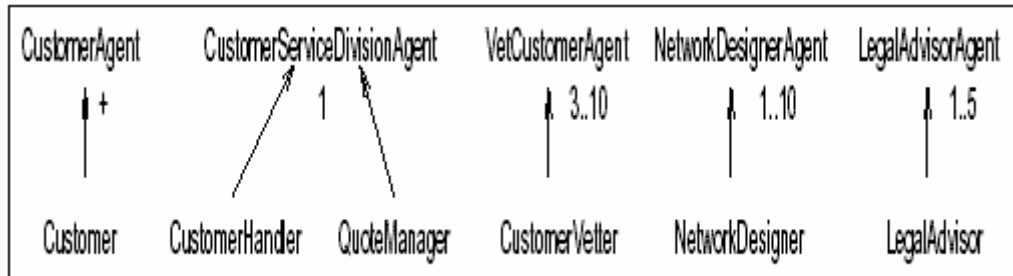


Figure 4: Agent Model for Business Process Management (Wooldridge, *et al.*, (1999))

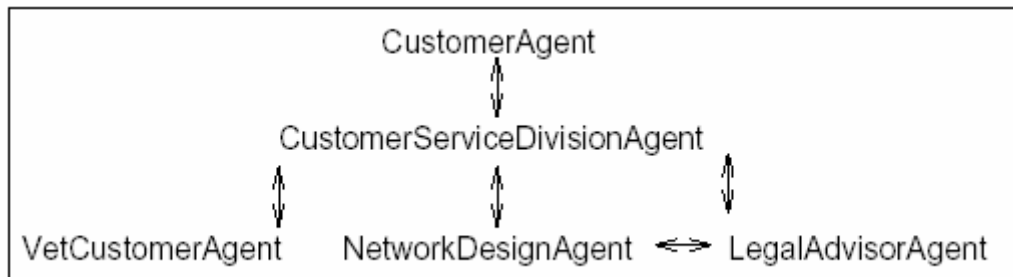


Figure 5: Acquaintance Model for Business Process Management (Wooldridge *et al.*, 1999)