Facilitating MAS Complete Life Cycle through the Protégé-Prometheus Approach

Marina V. Sokolova[1,2] and Antonio Fernández-Caballero[1]

[1] Universidad de Castilla-La Mancha, Escuela Politécnica Superior de Albacete & Instituto de Investigación en Informática de Albacete, 02071-Albacete, Spain
[2] Kursk State Technical University, Kursk, ul.50 Let Oktyabrya, 305040, Russia
smv1999@yandex.ru,caballer@dsi.uclm.es

Abstract. The approach of this paper aims to support the complete multi-agent systems life cycle, integrated by two existing and widely accepted tools, Protégé Ontology Editor and Knowledge-Base Framework, and Prometheus Development Kit. A general sequence of steps facilitating application creation is proposed in this paper. We propose that it seems reasonable to integrate all traditional software development stages into one single methodology. This view provides a general approach for MAS creation, starting with problem definition and resulting in program coding, deployment and maintenance. The proposal is successfully being applied to situation assessment issues, which has concluded in an agent-based decision-support system for environmental impact evaluation.

Keywords: Multi-agent systems, Software life cycle, Methodologies.

1 Introduction

Nowadays there are many works and approaches dedicated to multi-agent systems (MAS) development, which pay attention to internal MAS functionality, reasoning and its coding. Creation, deployment and post-implementation of MAS as software products is a complex process, which passes through a sequence of stages forming its life cycle [13][21]. Every step of the life cycle process has to be supported and provided by means of program tools and methodologies. In case of MAS development, in our opinion there is still no solution to a unified approach to cover all the stages. However, there are some previous works dedicated to this issue [2][12]. For instance, de Wolf and Holvoet [2] have introduced a methodology in the context of standard life cycle model, with accent to decentralization and macroscopic view of the process. The authors offer their approach on the assumption that the research task has already been defined, omitting the problem definition and domain analysis stages of MAS development process. But, a complete software development in case of MAS should be based on the following steps: (1) domain and system requirements analysis, (2) design, (3) implementation, (4) verification, and, (5) maintenance [9][12].

Some well known alternative agent-oriented software engineering methodologies, including MaSE [1] , Gaia [23], MASDK [6], Prometheus [15], Tropos [4],

N.T. Nguyen et al. (Eds.): KES-AMSTA 2008, LNAI 4953, pp. 63–72, 2008.

INGENIAS [5], among others, support some of the cited stages of MAS life cycle process. Nonetheless, these methodologies often work under the condition that the developer has already defined the problem and determined the goals and the tasks of the system. However, domain analysis is a crucial stage and has to be scrutinizingly examined and planned. Indeed, the whole deployed system functionality and efficiency depends on how precisely the problem was defined and the domain ontology was elaborated. In the most general case, when a MAS is distributed and has to deal with heterogeneous information, the domain analysis becomes even more important.

Therefore, it seems reasonable to integrate all the software development stages into one single methodology, which should provide a general approach to MAS creation, starting with the problem definition and resulting in program coding, deployment and maintenance. As a tool for the system and domain requirements, we suggest using an OWL-language based toolkit, as OWL has become a standard for ontologies description [14]. The Protégé Ontology Editor and Knowledge-Base Framework [16] complies a set of procedures for ontology creation and analysis, offering a set of plug-ins covering viewers, problem-solving methods, knowledge trees, converters, etc. According to our proposal, ontologies can be represented by means of Protégé and later may be incorporated into MAS. In order to provide the following stages with tools we have tested different methodologies. We came to the conclusion to use the Prometheus Development Tool (PDT) [17], which provides a wide range of possibilities for MAS planning and implementation: the system architecture, the system entities, their internals and communications within the system and with outer entities. The most important advantages of PDT are an easy understandable visual interface and the possibility to generate code for JACK^{TM} Intelligent Agents [11]. The proposal is summed up in Fig. 1.

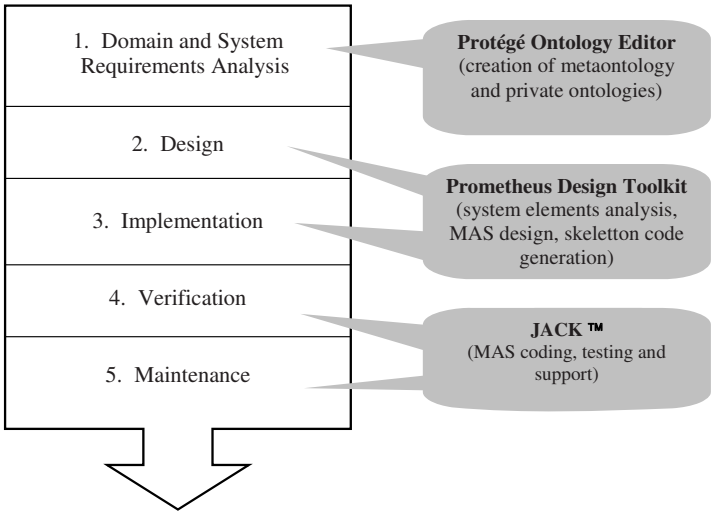

Fig. 1. The Protégé-Prometheus approach applied to the MAS life cycle

The paper is organized as follows. In section 2 the metaontology creation realized in Protégé is described and in section 3 the MAS designed in PDT is introduced. In section 4 our intention to implement the ideas for further usage of the integrated methodology are briefly explained.

2 Domain and MAS Requirements in Protégé

Ontology creation may be viewed as a crucial step in MAS design as it determines the system knowledge area and potential capabilities [7]. In the first part of this article a model of distributed metaontology that serves as a framework for MAS design is proposed. Its components - private ontologies - are described in extensive with respect to an application area and in terms of the used semantics.

When defining an ontology O in terms of an algebraic system, we have the following three attributes:

$$O = (C, R, \Omega) \tag{1}$$

where C is a set of concepts, R is a set of relations among the concepts, and Ω a a set of rules. The principal point of MAS is to determine the rules Ω and to evaluate them. Formula (2) proposes that the ontology for the domain of interest (or the problem ontology) may be described by offering proper meanings to C, R and Ω.

The model of the metaontology that we have created consists of five components, or private ontologies: the "Domain Ontology", the "Task Ontology", the "Ontology of MAS", the "Interaction Ontology" and the "Agent Ontology".

In first place, the "Domain Ontology", includes the objects of the problem area, the relations between them and their properties. It determines the components C and R of expression (2), which is detailed as:

$$OD =< I, C, P, V, Rs, Rl > \tag{2}$$

where the set C (see formula (2)) is represented by two components: Individuals (I) and Classes (C), which reflect the hierarchical structure of the objects of the problem area; P - are class properties; V - are the properties values; Rs - are values restrictions; Rl embodies the set R, and includes rules which state how to receive new individuals for the concrete class.

The "Task Ontology" contains information about tasks and respective methods, about the pre-task and post-task conditions, and informational flows for every task. The "Task Ontology" has the following model:

$$OT =< T, M, In, Ot, R > \tag{3}$$

where T is a set of tasks to be solved in the MAS, and M is a set of methods or activities related to the concrete task, In and Ot are input and output data flows, R is a set of roles that use the task. Component R is inherited from the "Ontology of MAS" through the property `belong to role`. The tasks are shared and accomplished in accordance with an order.

The "Ontology for MAS" architecture is stated as:

$$OA =< L, R, IF, Or > \tag{4}$$

where L corresponds to the logical levels of the MAS (if required), R is a set of determined roles, IF is a set of the corresponding input and output information represented by protocols. Lastly, the set Or determines the sequence of execution for every role (orders).

The interactions between the agents include an initiator and a receiver, a scenario and the roles taken by the interacting agents, the input and output information and a common communication language. They are stated in the "Interaction Ontology" as:

$$OI =< In, Rc, Sc, R, In, Ot, L > \tag{5}$$

Actually, as In and Rc Initiator and Receiver, respectively, of the interaction we use agents. The component Sc corresponds to protocols. R is a set of roles that the agents play during the interaction. In and Ot are represented by informational resources, required as input and output, respectively. Language L determines the agent communication language (ACL).

In our approach BDI agents [3], which are represented by the "Agent Ontology", are modeled. Hence, every agent is described as a composition of the following components:

$$Agent =< B, D, I > \tag{6}$$

Every agent has a detailed description in accordance with the given ontology, which is offered in a form of BDI cards, in which the pre-conditions and post-conditions of agent execution and the necessary conditions and resources for the agent successful execution are stated. Evidently, B, D and I stand for Believes, Desires and Intentions, respectively.

Metaontology is a specification for further MAS coding; it provides the necessary details about the domain, and describes system entities and functionality. It includes five components:

$$MO =< OD, OT, OA, OI, Agent > \tag{7}$$

where OD stands for the "'Domain Ontology"', OT for the "Task Ontology", OA "Ontology for MAS" architecture, OI is the "Interaction Ontology", and, $Agent$ is the "Agent Ontology".

Private ontologies mapping is made through slots of their components. So, the "Agent Ontology" has four properties:

1. `has intentions` - which contains individuals of the methods "M" class from the "Task Ontology".
2. `has believes` - which contains individuals from the "Domain Ontology".
3. `has desires` - which contains individuals from the "Task Ontology".
4. `has type` - which contains variables of *String* type.

There is a real connection between the "Task Ontology" and the "Domain Ontology". The *OT*, in turn, refers to the "Ontology of MAS" (*OA*), which is formally described by four components. The first two

- `level value`
- `order`

contain values of *Integer* type, which determine the logical level number and the order of execution for every role. Roles (*R*) are the individuals of the named ontology. The next two properties

- `has input`
- `has output`

refer to individuals of the "Interaction Ontology"; in particular, to protocols, which manage communications. Their properties are of type *String*:

- `has scenario`,
- `language`,
- `roles at scenario`.

The "Interaction Ontology" slots named `has initiator` and `has receiver` are the individuals of the "Agent Ontology" (*Agent*). Thus, agents are linked

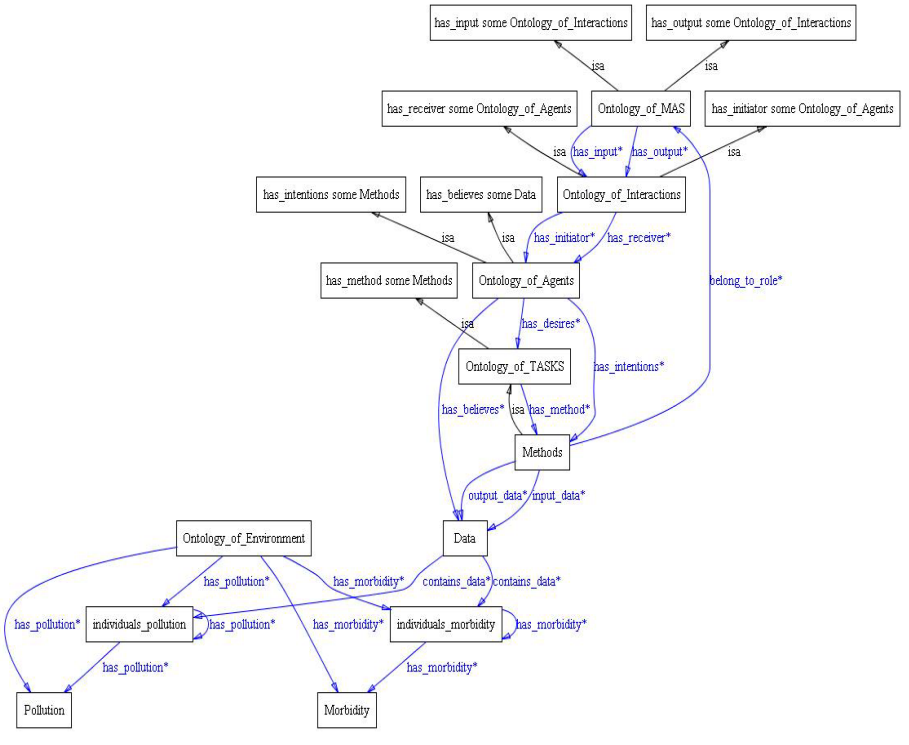

Fig. 2. Metaontology as a result of private ontologies mapping

to the proper protocols within the MAS. The *OD*, by means of its individuals - which contain data records - is connected to *Agent*, which uses the knowledge on the domain area as its believes. This way, the proposed metaontological model realized in Protégé covers the first four steps of the software development life cycle. The "System Elements Analysis" phase is covered by establishing the terminological basis for the further design.

3 System Design with Prometheus Development Tool

In order to validate the second step of our approach, we introduce a running example, consisting in an agent-based decision support system (ADSS) dedicated to monitoring environmental pollution information, analyzing this data, simulating with respect to health consequences, and making decisions for responsible system users [8][10].

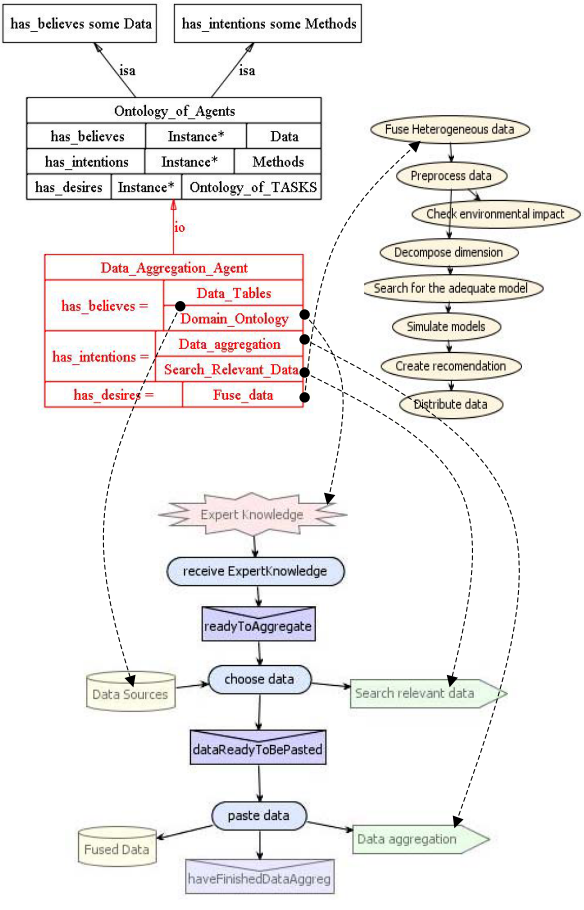

Fig. 3. An example of Protégé and Prometheus agent internals integration

Following the orientations described in section 2 for metaontology and private ontologies creation, the ADSS has been modeled as a logical three layer MAS architecture. The first layer is named **Information Fusion** and it acquires data from diverse sources, and preprocesses the initial information to be ready for further analysis. The second layer is named **Data Mining** and there are three roles at this level, dedicated to knowledge recovering through modeling, and calculation impact of various pollutants upon human health. The third layer, **Decision Making**, carries out a set of procedures including model evaluation, computer simulation, decision making and forecasting, based on the models created in the previous level. At every level of the system certain goals and tasks have to accomplished [18][19].

The system resembles a typical organizational structure. The agents are strictly dedicated to working with the stated sets of data sources. They solve the particular tasks and are triggered when all the necessary conditions are ful-

Table 1. Mapping between Protégé and Prometheus entities

Protégé Entity	**Prometheus Entity**	**Prometheus View**
"Domain Ontology"	Data	Data Coupling
"Task Ontology"		
Tasks	Goals	Goal Overview
Methods	Actions	System Roles
Input	Data	Data Coupling
Output	Data	Data Coupling
Roles	Roles	System Roles /Agent-Role Grouping
"Ontology of MAS" structure		
Levels	-	System Overview
Roles	Roles	System Overview /Agent-Role Grouping
Information Flows	Protocols	System Overview
Order	-	System Overview
"Interaction Ontology"		
Initiators	Agents	Agent Acquaintance
Receivers	Agents	Agent Acquaintance
Scenarios	Scenarios	Scenarios
Roles at Scenario	-	-
Input	Data	Data Coupling
Output	Data	Data Coupling
Language	-	-
"Agent Ontology"		
Believes	Data	Data Coupling
	Perceptions	Analysis Overview
Desires	Goals	Goal Overview
Intentions	Actions	System Overview /System Roles

filled, and there are positive messages from previously executed agents [22]. The system includes a set of roles, correlated with the main system functions and a set of agents related to each role. Actually, mostly every agent is associated to one role; only in case of "Function Approximation" role, there are two agents, one for data mining, and the other one for validation.

In Fig. 3 there is an illustration of the integration between metaontological concepts (and their properties) in Protégé and the Prometheus entities in the context of the agent internals. The "Data Aggregation agent" uses "Domain Ontology" and "Task Ontology", which are parts of the metaontology previously realized in Protégé. In order to closely analyze the integration of these methodologies, the mapping of the Protégé entities into Prometheus ones is shown in Table 1. For instance, Table 1 states similarities between entities of OD ("Domain Ontology") in Protégé and "Data" in Prometheus, which can be observed in the "Data Coupling diagram". The components of OT (see equation (3)) are converted into Prometheus entities and can be displayed as well. Some components of OA, such as *Levels* and *Order* do not have equivalents, as well as *Roles at Scenario* and *Language* components of OI, which serves more for a researcher during the MAS planning stage.

4 Conclusions

The integration of two existing and widely accepted tools, Protégé Ontology Editor and Knowledge-Base Framework, and Prometheus Development Kit, into a common methodology has been introduced in this paper. The Protégé Ontology Editor complies a set of procedures for ontology creation and analysis, offering a set of plug-ins such as viewers, problem-solving methods, knowledge trees, converters, and so on. To provide the following stages with tools, we have tested different methodologies, and finally decided to use the Prometheus Development Tool, which offers a wide range of possibilities for MAS planning and implementation, namely the system architecture, the system entities, their internals and communications within the system and with outer entities.

As the Prometheus methodology has been developed in collaboration with Agent Oriented Software, and a modified version of the Prometheus modeling language has been partially implemented in their JACK Intelligent Agents^{TM} development environment as a tool for visual modeling of the architectural design and plans, the next logical step of our approach is to implement under this environment. The JACK Design Tool is a software package for agent-based applications development in Java-based environment JACK.

Thus, the integrated approach covers all the stages of MAS planning and implementation, supporting them with tools and frameworks. The proposed fusion of methodologies, Protégé and Prometheus, was chosen because of the wide range of functions offered and their conformance to international standards. We believe that the common use of Protégé and Prometheus in complex developments would prevent researchers and developers from numerous misunderstandings. It should greatly help in domain requirements description, facilitating the complete

MAS development life cycle. However, other combinations of agent-oriented tools could be used, whenever it helps getting the same result and support during MAS development, deployment and maintenance.

Acknowledgements

Marina V. Sokolova is the recipient of Postdoctoral Scholarship 0000253836, Program II.E (Becas MAE) awarded by the Agencia Española de Cooperación Internacional of the Spanish Ministerio de Asuntos Exteriores y de Cooperación.

This work is supported in part by the Spanish Ministerio de Educación y Ciencia TIN2004-07661-C02-02 and TIN2007-67586-C02-02 grants, and the Junta de Comunida-des de Castilla-La Mancha PBI06-0099 grant.

References

1. DeLoach, S.A., Wood, M.F., Sparkman, C.H.: Multiagent systems engineering. International Journal of Software Engineering and Knowledge Engineering 11, 231–258 (2001)
2. de Wolf, T., Holvoet, T.: Towards a full life-cycle methodology for engineering decentralised multi-agent systems. In: Proceedings of the Fourth International Workshop on Agent-Oriented Methodologies, pp. 1–12 (2005)
3. Georgeff, M., Pell, B., Pollack, M., Tambe, M., Wooldridge, M.: The Belief-Desire-Intention model of agency. In: Rao, A.S., Singh, M.P., Müller, J.P. (eds.) ATAL 1998. LNCS (LNAI), vol. 1555, pp. 1–10. Springer, Heidelberg (1999)
4. Giunchiglia, F., Mylopoulos, J., Perini, A.: The Tropos software development methodology: Processes, models and diagrams. In: Third International Workshop on Agent-Oriented Software Engineering, pp. 162–173 (2002)
5. Gómez-Sanz, J., Pavon, J.: Agent oriented software engineering with INGENIAS. In: Mařík, V., Müller, J.P., Pěchouček, M. (eds.) CEEMAS 2003. LNCS (LNAI), vol. 2691, pp. 394–403. Springer, Heidelberg (2003)
6. Gorodetsky, V., Karsaev, O., Konushy, V., Mirgaliev, A., Rodionov, I., Yustchenko, S.: MASDK software tool and technology supported. In: International Conference on Integration of Knowledge Intensive Multi-Agent Systems, pp. 528–533 (2005)
7. Guarino, N., Giaretta, P.: Ontologies and knowledge bases: Towards a terminological clarification. In: Towards Very Large Knowledge Bases, pp. 25–32. IOS Press, Amsterdam (1995)
8. International Classification of Diseases (ICD) home page, `http://www.who.int/classifications/icd/en/`
9. ISO/IEC 12207 home page, `http://www.iso.org/iso/`
10. ISO 14031:1999 home page. Environmental management - Environmental performance evaluation - Guidelines, `http://www.iso.org/iso/`
11. Jack^{TM} Intelligent Agents home page, `http://www.agent-software.com/shared/home/`
12. Konichenko, A.V.: Distribution Information Systems Design Management. Rostov-Don Press, Russia (2005)
13. Maríık, V., McFarlane, D.: Industrial adoption of agent-based technologies. Intelligent Systems 20, 27–35 (2005)

14. OWL Web Ontology Language home page. (2004), http://www.w3.org/TR/owl-features/
15. Padgham, L., Winikoff, M.: Prometheus: A pragmatic methodology for engineering intelligent agents. In: Proceedings of the Workshop on Agent Oriented Methodologies (Object-Oriented Programming, Systems, Languages, and Applications), pp. 97–108 (2002)
16. Prometheus Design Tool home page, http://www.cs.rmit.edu.au/agents/pdt/
17. Protégé home page, http://protege.stanford.edu/
18. Sokolova, M.V., Fernández-Caballero, A.: A multi-agent architecture for environmental impact assessment: Information fusion, data mining and decision making. In: 9th International Conference on Enterprise Information Systems, ICEIS 2007, vol. AIDSS, pp. 219–224 (2007)
19. Sokolova, M.V., Fernández-Caballero, A.: An agent-based decision support system for ecological-medical situation analysis. In: Mira, J., Álvarez, J.R. (eds.) IWINAC 2007. LNCS, vol. 4528, pp. 511–520. Springer, Heidelberg (2007)
20. Schelfthout, K., Holvoet, T.: ObjectPlaces: an environment for situated multi-agent systems. In: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 1500–1501 (2004)
21. Vasconcelos, W.W., Robertson, D.S., Agusti, J., Sierra, C., Wooldridge, M., Parsons, S., Walton, C., Sabater, J.: A lifecycle for models of large multi-agent systems. In: Wooldridge, M.J., Weiß, G., Ciancarini, P. (eds.) AOSE 2001. LNCS, vol. 2222, pp. 297–318. Springer, Heidelberg (2002)
22. Weiss, G.: Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. The MIT Press, Cambridge (2000)
23. Wooldridge, M., Jennings, N.R., Kinny, D.: The Gaia methodology for agent-oriented analysis and design. Journal of Autonomous Agents and Multi-Agent Systems 3, 285–312 (2000)