Supporting multi-agent systems life cycle by integrating Protégé and Prometheus

Marina V. Sokolova

Universidad de Castilla-La Mancha,
Instituto de Investigación en Informática de Albacete,
Albacete, 02071, Spain
and
Kursk State Technical University,
Kursk, ul.50 Let Oktyabrya, 305040, Russia
E-mail: smv1999@yandex.ru

Antonio Fernández-Caballero*

Universidad de Castilla-La Mancha,
Instituto de Investigación en Informática de Albacete,
Albacete, 02071, Spain
E-mail: caballer@dsi.uclm.es
*Corresponding author

Abstract: In this paper, an approach aiming to support the complete multi-agent systems (MAS) life cycle is proposed. Two existing and widely accepted tools, Protégé Ontology Editor and Knowledge-Base Framework and Prometheus Development Kit, are integrated, offering a general sequence of steps facilitating application creation. It seems reasonable to integrate all traditional software development stages into one single methodology, which can provide a general approach for MAS creation, starting with problem definition and resulting in program coding, deployment and maintenance. The approach is successfully being applied to situation assessment issues, which has concluded in an agent-based decision-support system for environmental impact evaluation. An example is offered to evaluate the impact of environmental parameters upon human health in Spanish region Castilla-La Mancha, in general, and in the city of Albacete, in particular.

Keywords: ontologies; knowledge representation; agent-oriented software engineering; AOSE; Protégé; Prometheus.

Reference to this paper should be made as follows: Sokolova, M.V. and Fernández-Caballero, A. (2010) 'Supporting multi-agent systems life cycle by integrating Protégé and Prometheus', *Int. J. Intelligent Information and Database Systems*, Vol. 4, No. 3, pp.227–244.

Biographical notes: Marina V. Sokolova received her MSc in 2000 and her PhD thesis in 2004 at the Kursk State Technical University, Russia. Since 2004, she works as an Associate Professor at Kursk State Technical University. In 2006, she received a personal grant to conduct research at the University of Castilla-La Mancha, Spain. Her research interests include decision support systems, multi-agent systems, methods for data mining and knowledge discovery.

Antonio Fernández-Caballero received his degree in Computer Science from the Technical University of Madrid, Spain in 1993 and his PhD from the Department of Artificial Intelligence of the National University for Distance Education, Spain in 2001. Since 1995, he is an Associate Professor at the Department of Computer Science, University of Castilla-La Mancha, Spain. His research interests are in image processing, computer vision and agent technology. He has published more than 150 scientific papers. He is currently an Associate Editor of the *Pattern Recognition Letters* journal.

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 a MAS as a software product is a complex process, which passes through a sequence of stages forming its life cycle (Marík and McFarlane, 2005; Vasconcelos et al., 2001). 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 works dedicated to this issue (Gascueña and Fernández-Caballero, 2008; de Wolf and Holvoet, 2005; Konichenko, 2005). For instance, de Wolf and Holvoet (2005) have presented a methodology in the context of standard life cycle model, with accent to decentralisation 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 the MAS development process.

The software development in case of MAS is based on the following steps:

1 Domain analysis – is related to the analysis of the project idea, problem definition, extraction of aims, creation of goal trees, sequencing of tasks and subtasks to be solved. This stage also implies the domain ontology creation, which covers the problem area, the set of relations between the concepts and the rules to incorporate new knowledge. The experience of domain area experts is required on this stage.

2 Software elements analysis – this stage also deals with private ontologies creation; but now ontologies are created for the system and its elements. The sets of goals and tasks are related to the sets of system functions (roles), required resources (commonly in form of informational files), interactions and so on.

3 Specification – is the written description of the previous stages, which results in system meta-ontology creation.

4 Software architecture – implies the abstract representation of the system to meet the requirements. The software architecture includes interfaces for human-computer communication.

5 Implementation (coding) – the iterative process of program creation.

6 Testing – program testing under normal and/or critical conditions.

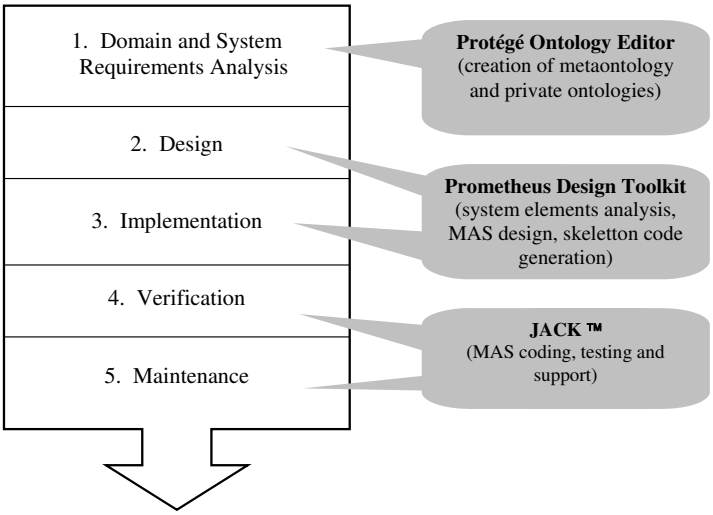
7 Deployment and maintenance – program application and support until the software is put into use. Sometimes some training classes on the software product are made.

8 End of maintenance – is the final stage of the software life cycle.

Some well-known alternative agent-oriented software engineering (AOSE) methodologies, including MaSE (DeLoach et al., 2001), Gaia (Wooldridge et al., 2000), MASDK (Gorodetsky et al., 2005), Prometheus (Padgham and Winikoff, 2002), Tropos (Giunchiglia et al., 2002) and INGENIAS (Gómez-Sanz and Pavón, 2003), among others, support some stages of the MAS life cycle process. Nonetheless, they 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 scrutinisingly 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 the MAS is distributed and has to deal with heterogeneous information, the domain analysis becomes even more important.

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 (see Figure 1). As a tool for the system and domain requirements, we suggest using an OWL-language-based toolkit, as OWL (2004) becomes a standard for ontologies description. The Protégé Ontology Editor and Knowledge-Base Framework (Protégé, 2008) complies a set of procedures for ontology creation and analysis, offering a set of plug-ins covering viewers, problem-solving methods, knowledge trees, converters and so on. According to our proposal, ontologies can be represented by means of Protégé and later may be incorporated into MAS.

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



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), which provides a wide range of possibilities for MAS planning and

implementation (PDT, 2008): 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 (2008).

2 Software tools for MAS life cycle support

2.1 MAS design: the Prometheus methodology

The Prometheus methodology defines a detailed process for specifying, designing and implementing agent-oriented software (AOS) systems. It consists of three phases (Padgham and Winikoff, 2004): the *system specification* phase, which focuses on identifying the goals and basic functionalities of the system, along with inputs (percepts) and outputs (actions) (Russell and Norvig, 1995); the *architectural design* phase, which uses the outputs from the previous phase to determine which agent types the system will contain and how they will interact; and the *detailed design* phase, which looks at the internals of each agent and how it will accomplish its tasks within the overall system.

System specification

The Prometheus methodology focuses particularly on specification of goals (van Lamsweerde, 2001) and on scenario description (Liu and Yu, 2001). In addition, it requires specification of functionalities – small chunks of behaviour – related to the identified goals. There is also a focus on how the agent system interfaces with the environment in which it is situated, in terms of percepts that arrive from the environmen, and actions that impact on the environment. As part of the interface specification, Prometheus also addresses interaction with any external data stores or information repositories. The aspects developed in the system specification phase are: specification of system goals with associated descriptors; development of a set of scenarios that have adequate coverage of the goals; definition of a set of functionalities that are linked to one or more goals and which provide a limited piece of system behaviour; and description of the interface between the agent system and the environment in which it is situated.

Goals are central to the functioning of the intelligent software agents that are going to realise the system. An initial brief system description provides a starting point for building an initial list of system goals. Then the goals are refined, and after asking ‘how this goal might be achieved?’ the answer gives the subgoals of the goal under consideration. A grouping of similar subgoals provides the basis for the functionalities.

Functionality is the term used for a chunk of behaviour, which includes a grouping of related goals, as well as percepts, actions and data relevant to the behaviour. Functionalities allow for a mixture of both top-down and bottom-up design. They are identified by a top-down process of goal development, and they provide a bottom-up mechanism for determining the agent types and their responsibilities.

Scenarios are complementary to goals in that they show the sequences of steps that take place within the system. Hence, scenarios are used primarily to illustrate the normal running of the system. As scenarios are developed, it becomes evident where there is a need for information from the environment and where actions are required. Possible scenario steps are achieving a goal, performing an action, receiving a percept or referring to another use case scenario.

Agent systems are typically situated in a changing and dynamic environment. The interface description firstly deals with the environment incoming information (percepts) and the mechanisms for affecting the environment (actions). At this stage, simply a list of percepts and actions is developed. Also, as scenarios and functionalities are developed, it is also important to note the data that is produced and used.

Architectural design

The three aspects that are developed during the *architectural design* phase are: deciding on the agent types used in the application; describing the interactions between agents using interaction diagrams and interaction protocols; and describing the system structure through the system overview diagram.

A major decision made during the architectural design is the types of agents used. Agent types are formed by combining functionalities using the criteria of coupling and cohesion. One strong reason for grouping functionalities together is that they use the same data and we do not want all agents to have to know about all other agents.

Once it has been decided on the agents in the system, the pathways of communication (which agent talks to which other agents) as well as the timing of communication (which messages are followed by which other messages) are identified. The communication is depicted explicitly in interaction diagrams, obtained after replacing any functionality with the agent that includes it and inserting a communication between agents where it is needed. It might be necessary to progress from interaction diagrams to protocols that define exactly which interaction sequences are valid within the system.

Finally, the interactions between the agents and the system interface in terms of percepts, actions and external data are specified. The overall design of the system is thus depicted in the system overview diagram, which brings all the items together.

Detailed design

In the *detailed design*, for each individual agent, it is decided what capabilities are needed for the agent to fulfil its responsibilities as outlined in the functionalities it contains. The process specifications to indicate more of the internal processing of the individual agents are developed. And when getting into greater detail, the capability descriptions to specify the individual plans, beliefs and events needed within the capabilities are developed. Then the views that show processing of particular tasks within individual agents are developed. It is during this final phase of detailed design that the methodology becomes specific to agents that use event-triggered plans in order to achieve their tasks.

As the design develops, what was originally a single functionality may well be split into smaller modules, each of which is a capability. For each capability, we need to determine the goals it is to achieve within the system. The agent overview diagram shows the relationships between the capabilities, thus providing a top level view of the agent internals. It also shows the message flow between the capabilities, as well as data internal to the agent.

In *detailed design*, we also want some mechanism to specify process as well as structure. For this, Prometheus uses a variant of UML activity diagrams (Fowler and Scott, 2003), where the activity within a single agent is specified, indicating interaction with other agents via the inclusion of messages within the diagram.

At the final step of the detailed design, each capability is broken down either into further capabilities or, eventually, into the set of plans that provide the details of how to react to situations or achieve goals. Capability diagrams take a single capability and describe its internals. Prometheus focuses on belief-desire-intention (BDI) platforms, which are characterised by a representation that has hierarchical plans with triggers and a description for each plan that indicates the context in which it is applicable. BDI systems choose among the plans that are applicable, backtracking to try another plan if the one initially chosen does not succeed.

At the lowest level of detail, each incoming message to the capability must have one or more plans that respond to that message. Each plan can typically be broken down into some number of subtasks, each of which is represented by an internal message.

Each plan is triggered by a specific event. This event may be the arrival of a percept, arrival of a message from another agent or an internal message or subtask within the agent. If there are several plans that could be triggered by a given event, then it is important to specify the conditions or situations under which the various plans are applicable. This is called context condition. A very important issue associated with the events is the precise specification of the information carried by that event.

2.2 MAS implementation, verification and maintenance: the JACK Agent Software

The JACK Agent Software Tool is a software package for agent-based applications development in Java-based environment JACK (JACK™ Intelligent Agents, 2008). JACK incorporates a visual interface, which supports application creation and can be created directly in JDK environment, or be imported, for example, from Prometheus, a graphical editor which provides agent systems design in accordance with its methodology. JACK enables building applications by providing a visual representation of the system components, in two modes: agent mode and team mode. There are several components of the AOS family, which are:

- JACK™ is the world's leading autonomous systems development platform. It has a proven track record in the development of applications that interact with a complex and ever-changing environment.
- JACKTeams™ supports the definition of autonomous teams. It supports a wide variety of teaming algorithms, allowing the representation of social relationships and coordination between team members.
- C-BDI™ implements the core of JACK's BDI in the Ada programming language. It is designed for applications where the software needs to be certificated, for example in onboard aviation systems.
- CoJACK™ is cognitive architecture used for modelling the variation in human behaviour. It is used in simulation systems to underpin virtual actors.
- JAE-Box™ can be thought of as 'autonomy in a box'. It is a JACK-based ruggedises single-board computer.
- Surveillance Agent™ is a JACK-based product that assists surveillance and intelligence agencies in the analysis of behaviour patterns.

JACK, written in Java, provides object-oriented programming for the system, encapsulating the desired behaviour in modular units so that agents can operate

independently. JACK intelligent agents are based on the BDI model, where autonomous software components where the agent pursues its given goals (desires), adopting the appropriate plans (intentions) according to its current set of data (beliefs) about the state of the world.

Hence, a JACK agent is a software component that has:

- a set of beliefs about the world (its dataset)
- a set of events that it will respond to
- a set of goals that it may desire to achieve (either at the request of an external agent, as a consequence of an event or when one or more of its beliefs change)
- a set of plans that describe how it can handle the goals or events that may arise.

JACK permits the creation of multiple autonomous agents, which can execute in agent and in team mode within an MAS. MAS creation can be realised using graphical interface.

Each JACK project file includes design views, agent and data model of the application. Agent model has containers to determine and code agents and their capabilities, plans, event types and belief structures. The JACK Agent Language is a superset of Java – it contains the Java components and syntax while extending it with constructs to represent agent-oriented features. The JACK Agent Compiler preprocesses JACK Agent Language source files and converts them into pure Java, which can then be compiled into Java virtual machine code. The JACK Agent Kernel is the runtime engine for programs written in the JACK Agent Language, which provides a set of agent-oriented classes. JACK extension to team mode permits teams models to be treated as peers and introduces new concepts as team, role, teamdata and teamplan, which are required to widen the semantics of some elements and to accept team reasoning entity, knowledge and internal coordination of the agents within the team.

The key concept that appears here is the role concept. A role defines the means of interacting between a containing team (a role tenderer) and a contained team (a role performer or role filler). In JACK Team mode each team has its lifetime, which is divided into two phases: first phase is for setting up an initial role obligation structure; and the second phase constitutes the actual operation of the team. In addition to the agent believes, in team mode, knowledge can be 'propagated' over the team members.

Thus, Prometheus Development Kit permits the creation of the skeleton code for its later implementation in JACK, which facilitates the stages of MAS planning and coding. Actually, JACK teams can be used for complex distributed system modelling and problem solving.

3 Our approach to MAS life cycle support

3.1 Domain and system 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 (Guarino and Giaretta, 1995). In this section, a model of distributed meta-ontology that serves as a framework for MAS design is proposed (see Figure 2). 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 set of rules. The principal point of MAS is to determine the rules Ω and to evaluate them. Formula (1) 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 meta-ontology 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, relations between them and their properties. It determines the components C and R of expression (1), which is detailed as:

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

where the set C [see formula (1)] 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; and 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 of 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; and 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, which are represented by the 'agent ontology', are modelled. Hence, every agent is described as a composition of the following components (Georgeff et al., 1998):

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

Every agent has a detailed description in accordance with the given ontology, which is offered in a form of BDI cards, in which the preconditions and postconditions 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.

Now, the meta-ontology 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 of MAS' architecture; OI is the 'interaction ontology', $Agent$ 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`

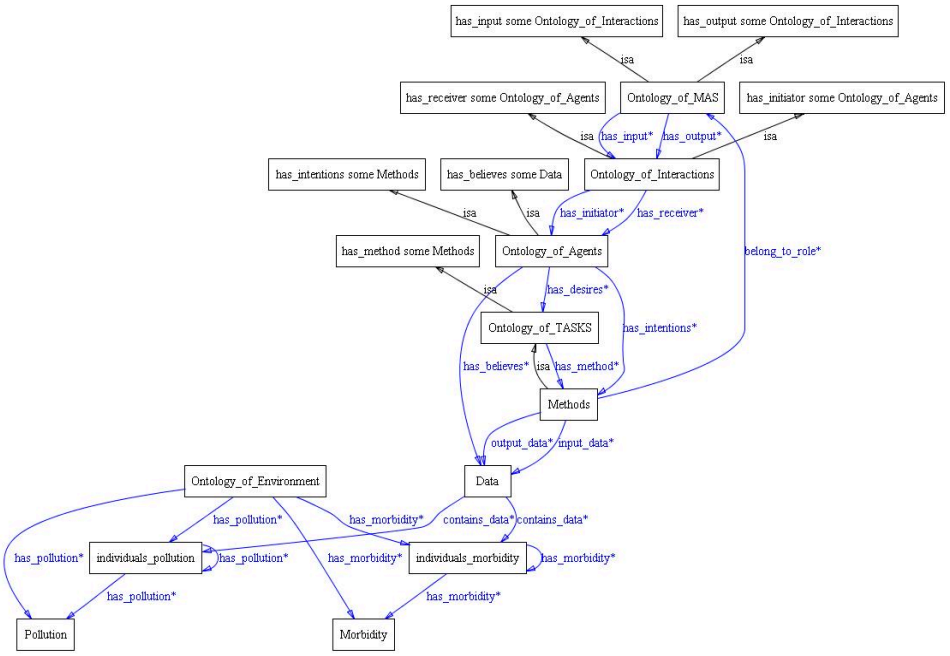
refer to individuals of the 'interaction ontology'; in particular to protocols, which manage communications. Their properties are of type *string*:

- `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 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 meta-ontological model realised 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.

Figure 2 Meta-ontology as a result of private ontologies mapping (see online version for colours)



3.2 System design with PDT

AOS development is one of the recent contributions to the field of software engineering. To date, numerous methodologies for AOS development have been proposed in the literature. However, their application to real world problems is still limited due to their lack of maturity. Evaluating their strengths and weaknesses is an important step towards developing better methodologies in the future.

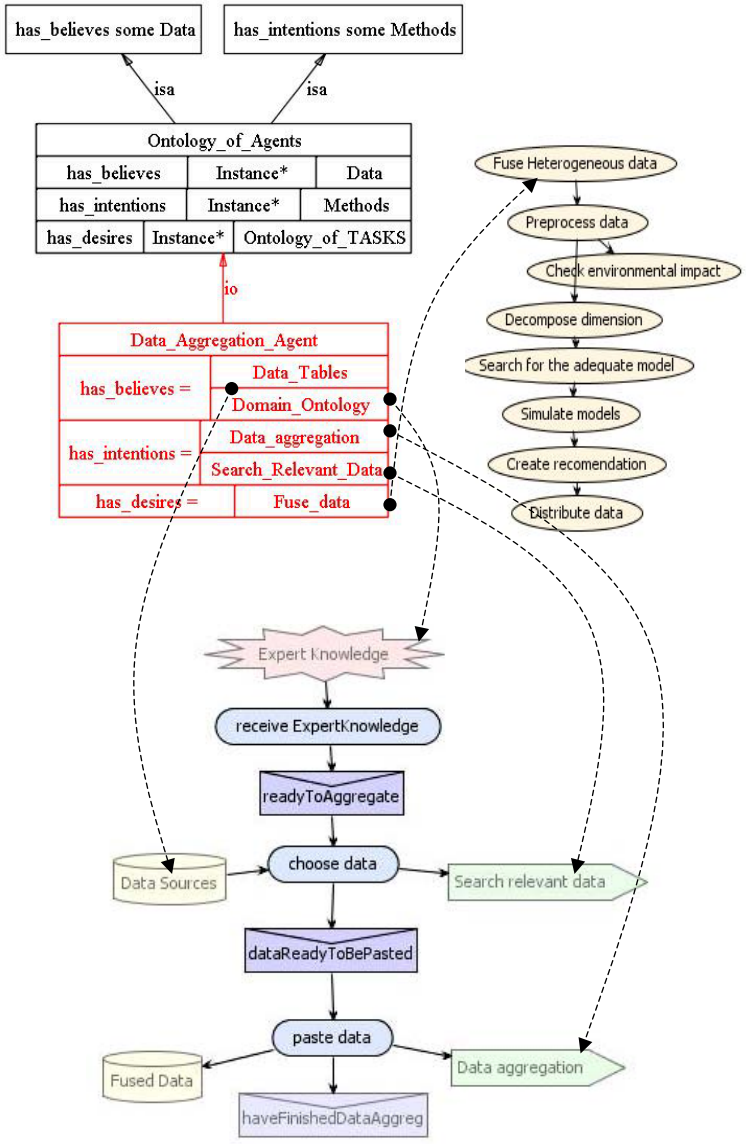
MAS brings some difficulties to a researcher, which are caused by task identifications, specifying sets of protocols, interactions, methods and agents behaviours. That makes software design tools more sophisticated, which operate with new concepts as agents, goals, tasks, interactions, plans, believes, etc. Methodologies offer different tools to cope with the complicity and facilitate MAS planning and design (Bergenti et al., 2004).

In order to validate the second step of our approach, we introduce a running example, consisting in an agent-based decision support system (ADSS) (Sokolova

and Fernández-Caballero, 2009) dedicated to monitoring environmental pollution information, analysing this data, simulating with respect to health consequences and making decisions for responsible system users (ICD, 2008; ISO 14031, 1999).

The MAS is a logical three-layer architecture. The first level 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 modelling and calculation impact of various pollutants upon human health. The third level, *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 be accomplished (e.g., Sokolova and Fernández-Caballero, 2007a, 2007b).

Figure 3 Integrating Protégé and Prometheus agent internals (see online version for colours)



The system resembles a typical organisational structure The agents are strictly dedicated to work with the stated sets of data sources. They solve the particular tasks and are triggered when all the necessary conditions are fulfilled, and there are positive messages from previously executed agents (Weiss, 2000). 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 Figure 3, there is an illustration of the integration between meta-ontological concepts and their properties from 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 meta-ontology previously realised in Protégé. In order to closely analyse the integration of these methodologies, the mapping of the Protégé entities into Prometheus ones is shown (see Table 1).

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

Table 1 states similarities between entities of OD in Protégé and data in Prometheus, which can be seen in the data coupling diagram. The components of the OT [see equation (3)] are converted into Prometheus entities and can be displayed as well. Some components of the OA, such as *levels* and *order* do not have some equivalents, as well as *roles at scenario* and *language* components of the OI, which serves more for a researcher during the MAS planning stage.

3.3 System implementation in JACK Software tool

The ADSS has an open agent-based architecture, which would allow us an easy incorporation of additional modules and tools, enlarging a number of functions of the system. The system belongs to the organisational type, where every agent obtains a class of tools and knows how and when to use them. Actually, such types of systems have a planning agent, which plans the orders of the agents' executions. In our case, the main module of the Jack program carries out these functions. In Figure 4 a part of the code is shown. There, the Data Aggregation agent is constructed with a constructor:

```
DataAggregationAgent DAA1 = new DataAggregationAgent ("DAA")
```

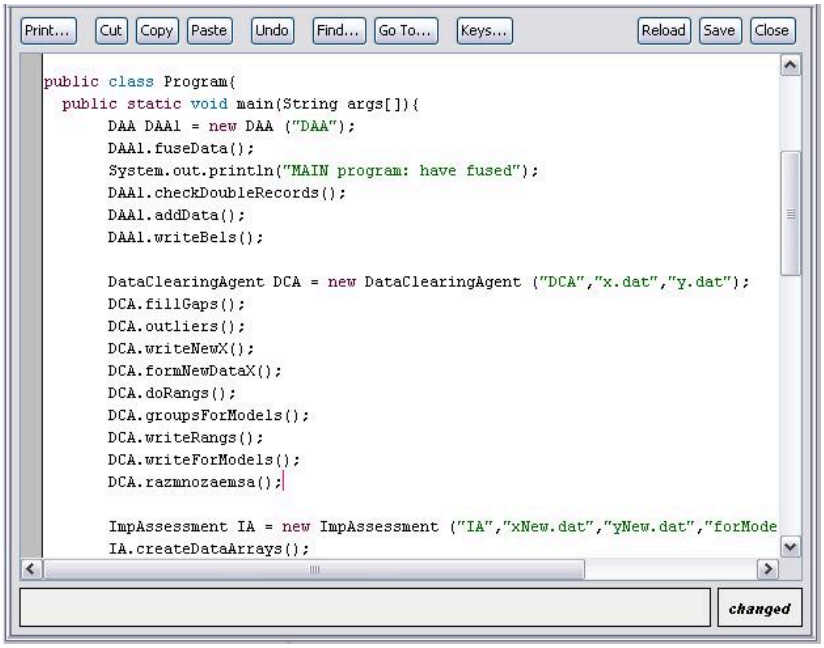
and then some of its methods are called, for example, DAA1.fuseData(). The DataClearingAgent is constructed as:

```
DataClearingAgent DCA = new DataClearingAgent("DCA", "x.dat", "y.dat")
```

where '`x.dat`' and '`y.dat`' are agents beliefs of 'global' type. This means that they are open and can be used by the other agents within the system. Finally, the ViewAgent, which displays the outputs of the system functionality and realise interaction with the system user, is called.

As the system is autonomous and all the calculations are executed by it, the user has only access to the result outputs and the simulation window. He/she can review the results of impact assessment, modelling and forecasting and try to simulate tendencies by changing the values of the pollutants.

Figure 4 The main program window in JACK (see online version for colours)



As the system is autonomous and all the calculations are executed by the proper system, the user has only access to the result outputs and the simulation window. He/she can review the results of impact assessment, modelling and forecasting and try to simulate tendencies by changing values of the pollutants.

4 Simulation results

The MAS has an open agent-based architecture, which allows an easy incorporation of additional modules and tools, thus enlarging a number of functions of the system. The system belongs to the organisational type, where every agent obtains a class of tools and knows *how* and *when* to use them. Actually, such types of systems have a planning agent, which plans the orders of the agents' executions. In our case, the main module of the JACK^{TM} program carries out these functions. The ViewAgent displays the outputs of the system functionality and realises the interaction with the system user.

To evaluate the impact of environmental parameters upon human health in Castilla-La Mancha (a Spanish region), in general, and in the city of Albacete in particular, we have collected retrospective data since year 1989, using open information resources offered by the Spanish Institute of Statistics and by the Institute of Statistics of Castilla-La Mancha. As indicators of human health and the influencing factors of environment, which can cause negative effect upon the noted above indicators of human health, the factors described in Table 2 were taken.

Table 2 Diseases studied in our research

Type of disease/pollutant	*Disease class*
1 Endogenous diseases	Certain conditions originating in the prenatal period
	Congenital malformations, deformations and chromosomal abnormalities
2 Exogenous diseases	Certain infectious and parasitic diseases
	Neoplasm
	Diseases of the blood and blood-forming organs and certain disorders involving the immune mechanism
	Endocrine, nutritional and metabolic diseases
	Mental and behavioural disorders
	Diseases of the nervous system
	Diseases of the eye and adnexa
	Diseases of the ear and mastoid process
	Diseases of the circulatory system
	Diseases of the respiratory system
	Diseases of the digestive system
	Diseases of the skin and subcutaneous tissue
	Diseases of the musculoskeletal system and connective tissue
	Diseases of the genitourinary system
	Pregnancy, childbirth and the puerperium
	Symptoms, signs and abnormal clinical and laboratory findings, not elsewhere classified
	External causes of morbidity and mortality

The ADSS has recovered data from plain files, which contained the information about the factors of interest and pollutants and fused in agreement with the ontology of the problem area. It has supposed some necessary changes of data properties (scalability,

etc.) and their preprocessing. After these procedures, the number of pollutants valid for further processing has decreased from 65 to 52. This significant change was caused by many blanks related to several time series, as some factors have started to be registered recently. After considering this as an important drawback, it was not possible to include them into the analysis. The human health indicators, being more homogeneous, have been fused and cleared successfully.

The impact assessment has shown the dependencies between water characteristics and neoplasm, complications of pregnancy, childbirth and congenital malformations, deformations and chromosomal abnormalities. Within the most important factors apart from water pollutants, there are indicators of petroleum usage, mines outcome products and some types of wastes. Table 3 shows that within the most important factors apart from water pollutants, there are indicators of petroleum usage, mines output products and some types of wastes.

Table 3 Part of the table with the outputs of impact assessment

Disease class	*Pollutant, which influence upon the disease*
1 Neoplasm	Nitrites in water; miner products; DBO5; asphalts; dangerous chemical wastes; fuel-oil; petroleum liquid gases; water: solids in suspension; non-dangerous chemical wastes
2 Diseases of the blood and blood-forming organs, the immune mechanism	DBO5; miner products; fuel-oil; nitrites in water; dangerous wastes of paper industry; water: solids in suspension; dangerous metallic wastes
3 Pregnancy, childbirth and the puerperium	Kerosene; petroleum; petroleum autos; petroleum liquid gases; gasohol; fuel-oil; asphalts; water: DQO; DBO5; solids in suspension; water: nitrites
4 Certain conditions originating in the prenatal period	Non-dangerous wastes: general wastes; mineral, constriction, textile, organic, metal wastes; dangerous oil wastes
5 Congenital malformations, deformations and chromosomal abnormalities	Gasohol; fuel-oil; DQO in water; producing asphalts; petroleum; petroleum autos; kerosene; petroleum liquid gases; water: DBO5, nitrites; water: solids in suspension

The ADSS has a wide range of methods and tools for modelling, including regression, neural networks, group method of data handling (GMDH) and hybrid models. The function approximation agent selected the best models, which were: simple regression – 4,381 models; multiple regression – 24 models; neural networks – 1,329 models; and GMDH – 2,435 models. The selected models were included into a committee machine. We have forecasted diseases and pollutants values for the period of four years, with a six month step and visualised their tendencies which, in common, and in agreement with the created models, are going to overcome the critical levels. The control of the 'significant' factors, which cause impact upon health indicators, could lead to the decrease of some types of diseases.

Committee machines provide universal approximation, as the responses of several predictors (experts) are combined by means of a mechanism that does not involve the input signal and the ensemble average value is received. As predictors, we used regression and neural network-based models. An example of the final model, received by committee machine for the 'neoplasms' for the region of Castilla-La Mancha, Spain is provided in Figure 5, and the results of the impact assessment are shown in Figure 6.

Figure 5 The final model for one of the diseases of the case study (see online version for colours)

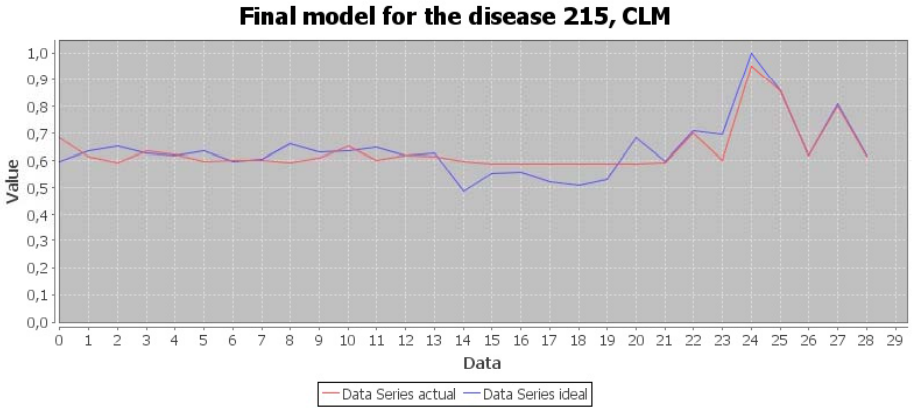
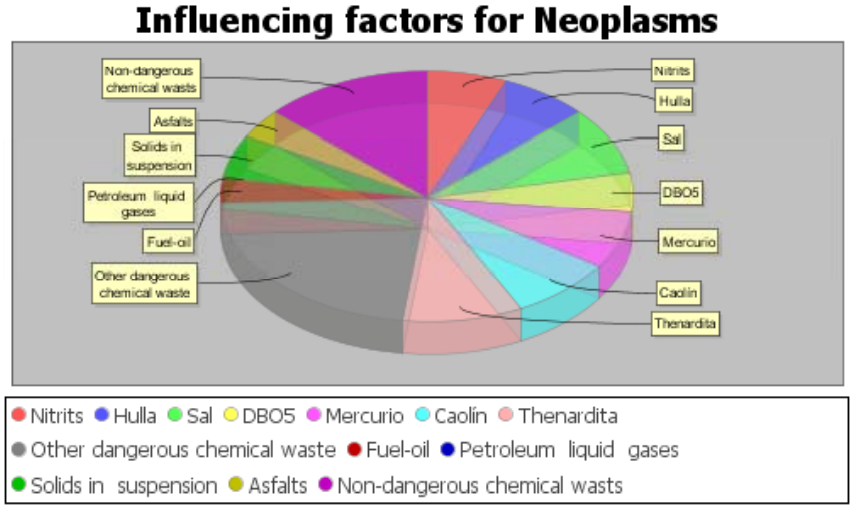


Figure 6 The outcomes of the impact assessment for one of the diseases of the case study (see online version for colours)



5 Conclusions

Our particular approach to support the complete MAS life cycle has been described and tested in this paper. 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 depth. 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 PDT, 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 AOS, and a modified version of the Prometheus modelling language has been partially implemented in the JACK™ Intelligent Agents development environment as a tool for visual modelling 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

This work is supported in part by the Spanish Ministerio de Ciencia e Innovación TIN2007-67586-C02-02 grant and the Junta de Comunidades de Castilla-La Mancha PII2I09-0069-0994, PII2I09-0071-3947 and PEII09-0054-9581 grants.

References

Bergenti, F., Gleizes, M.P. and Zambonelli, F. (2004) *Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook*, Springer.

de Wolf, T. and Holvoet, T. (2005) 'Towards a full life-cycle methodology for engineering decentralised multi-agent systems', *The Fourth International Workshop on Agent-Oriented Methodologies*, pp.1–12.

DeLoach, S.A., Wood, M.F. and Sparkman, C.H. (2001) 'Multiagent systems engineering', *International Journal of Software Engineering and Knowledge Engineering*, Vol. 11, pp.231–258.

Fowler, M. and Scott, K. (2003) *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, Addison Wesley.

Gascueña, J.M., Fernández-Caballero, A. (2008) 'Prometheus and INGENIAS agent methodologies: a complementary approach', *The Seventh International Conference on Autonomous Agents and Multi-Agent Systems, Workshop 3: Agent-Oriented Software Engineering*.

Georgeff, M., Pell, B., Pollack, M., Tambe, M. and Wooldridge, M. (1998) 'The belief-desire-intention model of agency', *Intelligent Agents V: Agent Theories, Architectures and Languages, Lecture Notes in Computer Science*, Vol. 1555, pp.1–10.

Giunchiglia, F., Mylopoulos, J. and Perini, A. (2002) 'The Tropos software development methodology: processes, models and diagrams', *Third International Workshop on Agent-Oriented Software Engineering*, pp.162–173.

Gómez-Sanz, J. and Pavón, J. (2003) 'Agent oriented software engineering with INGENIAS', *International Central and Eastern European Conference on Multi-Agent Systems, Lecture Notes in Computer Science*, Vol. 2691, pp.394–403.

Gorodetsky, V., Karsaev, O., Konushy, V., Mirgaliev, A., Rodionov, I. and Yustchenko, S. (2005) 'MASDK software tool and technology supported', *International Conference on Integration of Knowledge Intensive Multi-Agent Systems*, pp.528–533.

Guarino, N. and Giaretta, P. (1995) 'Ontologies and knowledge bases: towards a terminological clarification', *Towards Very Large Knowledge Bases*, pp.25–32.

International Classification of Diseases (ICD) (2008) Available at http://www.who.int/classifications/icd/en/.

ISO 14031 (1999) 'Environmental management – environmental performance evaluation – guidelines', available at http://www.iso.org/iso/.

Jack^{TM} Intelligent Agents (2008) Available at http://www.agent-software.com/shared/home/.

Konichenko, A.V. (2005) *Distribution Information Systems Design Management*, Rostov-Don Press, Russia.

Liu, L. and Yu, E. (2001) 'From requirements to architectural design: using goals and scenarios', *From Software Requirements to Architectures*, pp.22–30.

Marík, V. and McFarlane, D. (2005) 'Industrial adoption of agent-based technologies', *Intelligent Systems*, Vol. 20, pp.27–35.

Padgham, L. and Winikoff, M. (2002) 'Prometheus: a pragmatic methodology for engineering intelligent agents', *Workshop on Agent Oriented Methodologies (Object-Oriented Programming, Systems, Languages and Applications)*, pp.97–108.

Padgham, L. and Winikoff, M. (2004) *Developing Intelligent Agent Systems: A Practical Guide*, John Wiley & Sons.

Prometheus Design Tool (2008) Available at http://www.cs.rmit.edu.au/agents/pdt/.

Protégé, available at http://protege.stanford.edu/.

Russell, S. and Norvig, P. (1995) *Artificial Intelligence: A Modern Approach*, Prentice Hall.

Sokolova, M.V. and Fernández-Caballero, A. (2007a) 'A multi-agent architecture for environmental impact assessment: information fusion, data mining and decision making', *9th International Conference on Enterprise Information Systems*, pp.219–224.

Sokolova, M.V. and Fernández-Caballero, A. (2007b) 'An agent-based decision support system for ecological-medical situation analysis', *2nd International Work-conference on the Interplay between Natural and Artificial Computation, Lecture Notes in Computer Science*, Vol. 4528, pp.511–520.

Sokolova, M.V. and Fernández-Caballero, A. (2009) 'Modeling and implementing an agent-based environmental health impact decision support system', *Expert Systems with Applications*, Vol. 36, pp.2603–2614.

van Lamsweerde, A. (2001) 'Goal-oriented requirements engineering: a guided tour', *The 5th IEEE International Symposium on Requirements Engineering*, pp.249–263.

Vasconcelos, W.W., Robertson, D.S., Agusti, J., Sierra, C., Wooldridge, M., Parsons, S., Walton, C. and Sabater, J. (2001) 'A lifecycle for models of large multi-agent systems', *The Second International Workshop on Agent-Oriented Software Engineering, Lecture Notes in Computer Science*, Vol. 2222, pp.297–318.

Web Ontology Language (OWL) (2004) Available at http://www.w3.org/TR/owl-features/.

Weiss, G. (2000) *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, The MIT Press.

Wooldridge, M., Jennings, N.R. and Kinny, D. (2000) 'The Gaia methodology for agent-oriented analysis and design', *Journal of Autonomous Agents and Multi-Agent Systems*, Vol. 3, pp.285–312.