

A CASE tool supported methodology for the design of multi-agent systems

Massimo Cossentino, Colin Potts

Abstract— The design of multi-agent systems is different from similar problems because the concept of agent involves the notions of autonomy and intelligence. As a consequence agent-based software engineering approaches must learn from classical design approaches but should go further introducing an explicit representation of the previous cited notions as well as of ontology, communications, mobility and other agents related issues.

Even if these arguments are more than sufficient to justify the study of specific approaches we also think that today a successful design methodology should include some other strategic factors: the use of a standard and well known design language (like UML), the support of a specific CASE tool to simplify the work of the designer and the attention for the automatic production of large parts of code.

We propose PASSI (Process for Agent Societies Specification and Implementation) as a solution to the above arguments. It comprehends the construction of five models (System Requirements, Agent Society, Agent Implementation, Code Model and Deployment Model) which include several distinct phases. We also illustrates the contribute of an add-in that we have produced for a commercial UML-based CASE tool in order to have a dedicated design environment that proves more productive of the general purpose ones.

Index Terms— CASE tools, Design Methodology, Multi-Agent Systems, Software Engineering.

I. INTRODUCTION

SEVERAL works can be found in literature about the design and representation of multi-agents systems [21][22][23][24][25][26]. Some approaches propose representations involving abstractions of social phenomena and knowledge [21][25][26] obtaining an expressive representation of these aspects but maintaining a distance from the implementation level that introduces a difficulty in the production of the final code solution. On the contrary some others maintain an high level of attention both for analysis steps and implementation issues but are less interested in the representation of the social aspects of the MAS [22][23][24].

Massimo Cossentino is with the CERE/CNR (Centro di Studi sulle Reti di Elaboratori-Consiglio Nazionale delle Ricerche), V.le delle Scienze c/o Centro Univ. Calcolo, 90128 Palermo Italy. (phone: +39-091.6566274; fax: +39.0916529124; e-mail: cossentino@cere.pa.cnr.it).

Colin Potts is with the College of Computing of the Georgia Institute of Technology, Atlanta (GA) 30332-0280 USA (e-mail: potts@cc.gatech.edu).

We think that multi-agent systems (MAS) differ from non-agent based systems because agents are intended to be autonomous units of intelligent functionality. As a consequence, agent-based software engineering methods must complement standard design activities and representations with models of the agent society.

We give also a great importance for the success of a design methodology to some strategic factors: the use of a standard and well known design language (like UML), the support of a specific CASE tool to simplify the work of the designer and the attention for the automatic production of large parts of code (in order to increase the productivity and reduce the number of human errors).

We propose PASSI (Process for Agent Societies Specification and Implementation) as a solution to the above arguments. This methodology is the result of a long period of study [1][2][3] and experimentation mainly in robotics [4][5]. It is composed of five models (System Requirements, Agent Society, Agent Implementation, Code Model and Deployment Model) which include several distinct phases. We also produced an add-in for a diffused commercial UML-based CASE tool (Rational Rose) in order to have a dedicated design environment that proves more productive of the general purpose ones. The code production phase is also strongly supported by the automatic generation of a great amount of code. This is possible thanks to the simplicity of the structure of the FIPA architecture [27] that we assume as a reference for our approach, to the use of an XML content language for the messages between the agents (so that the content of each message can be straightforwardly derived from the design) and because of a library of reusable patterns of code.

The following sections are organized as follows: in section two we provide a quick overview of the PASSI methodology, in section three we discuss the different phases of PASSI illustrating the contribute of the CASE tool add-in using a simple example coming from robotics, in section four we quickly review the main themes of the paper and address some of the future work issues.

II. AN OVERALL VIEW OF THE DESIGN METHODOLOGY

Several design methodologies have been proposed in literature (Gaia [26], MASE [9], CASSIOPEIA [28],...) but they didn't satisfy some of the needs coming from our experiments with multi-agents systems applied to robotics. Several of the previous methodologies are lacking of

implementation level design support, some others not use a standard notation like UML or use a design philosophy that is far from the common experience of the greatest part of software engineers who are usually skilled with object-oriented approaches. Another important aspect that we difficulty found in other methodologies was a specific attention towards code reuse or patterns.

A definition of what is an agent in our approach can be helpful for the explanation of PASSI. We think at an agent as the instance of an agent class that is the software implementation of an autonomous entity capable of pursuing an objective through its autonomous decisions, actions and social relationships. An agent may occupy several functional roles to achieve its goals. A role is the function temporarily assumed by the agent in the society while pursuing a sub-goal. During this activity the agent uses one or more of its tasks. A task is a series of elementary pieces of behavior (actions) necessary for a specific purpose. Each task carries out one of the agent's decisions/actions/social relationships.

In this definition we can find the concepts of agent, role, task and action. These are some of the elements that will compose our system. In the following we will discuss the way we use to define these component to obtain the desired behavior.

In order to better understand our approach it can be useful to specify that the implementation of each agent will be done using a class derived from the base-agent type of the chosen platform; the tasks will be implemented as subclasses of the agent-class and the actions are methods of these classes. In this perspective, a role will be the result of a series of behaviors realized by the actions of several different tasks of the same agent.

We will illustrate the methodology with an example coming from robotics: we will design a multi-agent system in order to obtain some specific behaviors from a robot provided with video, IR and odometry sensors.

We will illustrate the methodology with an example coming from robotics: we will design a multi-agent system in order to obtain some specific behaviors from a robot provided with video, IR and odometry sensors.

The models and phases of PASSI are:

1. **System Requirements Model.** A model of the system requirements in terms of agency and purpose.

It is composed of four phases: (a) Domain Description (D.D.): A functional description of the system using conventional use-case diagrams. (b) Agent Identification (A.Id.): The phase of attribution of responsibility to agents, represented as stereotyped UML packages. (c) Role Identification (R.Id.): A series of sequence diagrams exploring the responsibilities of each agent through role-specific scenarios. (d) Task Specification (T.Sp.): Specification of the capabilities of each agent with activity diagrams.

2. **Agent Society Model.** A model of the social interactions and dependencies among the agents involved in the solution. Developing this model involves three steps in addition to part of the previous model: (a) Role Identification (R.Id.): See the System Requirements Model. (b) Ontology Description (O.D.): Use of class diagrams and OCL constraints to describe the knowledge ascribed to individual agents and the pragmatics of their interactions. (c) Role Description (R.D.). Class diagrams are used to show the roles played by agents, the tasks involved, communication capabilities and inter-agent dependencies. (d) Protocol Description (P.D.). Use of sequence diagrams to specify the grammar of each pragmatic communication protocol in terms of speech-act performatives.
3. **Agent Implementation Model.** A classical model of the solution architecture in terms of classes and methods, the most important difference with common Object-oriented approach is that we have two different levels of abstraction, the social (multi-agent) level and the single-agent level. This model is composed of the following steps: (a) Agent Structure Definition (A.S.D.): Conventional class diagrams describe the structure of solution agent classes. (b) Agent Behavior Description (A.B.D.): Activity diagrams or statecharts describe the behavior of individual agents.
4. **Code Model.** A model of the solution at the code level requiring the following steps to produce: (a) Generation of code from the model using one of the functionalities of the PASSI add-in. It is possible to generate not only the skeletons but also largely reusable parts of the methods implementation based

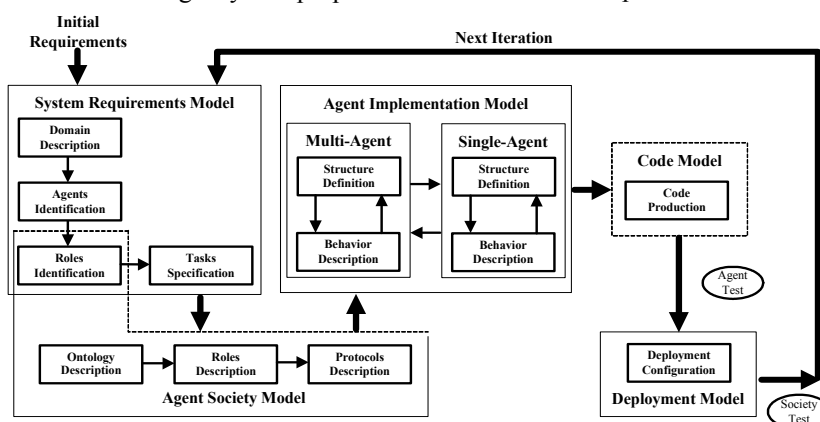


Fig. 1. The models and phases of the PASSI methodology

on a library of code and associated design descriptions. (b) Manual completion of the source code.

5. **Deployment Model.** A model of the distribution of the parts of the system across hardware processing units, and their migration between processing units. It involves one step: Deployment Configuration (D.C.); deployment diagrams describe the allocation of agents to the available processing units and any constraints on migration and mobility.

Testing: the testing activity has been divided into two different steps: the single-agent test is devoted to verifying the behavior of each agent regarding the original requirements for the system solved by the specific agent. During the society test, it is carried on the validation of the overall results of this iteration and of the integration of the different agents.

In the PASSI methodology we can identify several iterations. The requirements increment/evolution iteration is used to incrementally design/implement the software. It is a common practice and it proved useful also in agent-based software. The multi-agent/single-agent iteration (in the agent implementation model) describes the dependencies between the multi-agent level (where each agent interacts with the others) and the single-agent level (where the focus is the structure and behavior needed for each specific agent).

III. THE PHASES OF PASSI

A. Domain Description Phase

Both in object-oriented [6][7][8] and agent-oriented [9] design practice, it is common, during requirements analysis, to focus on the identification of the system goals. In PASSI we chose a different approach. The requirements are expressed in terms of use-case diagrams using classical object-oriented methods [10][11] or through the informal application of a scenario-based method such as GBRAM [7] or ScenIC [12]. The result is a hierarchical series of use-case diagrams where scenarios could be detailed using sequence diagrams.

B. Agent Identification Phase

Very different methods are proposed in literature in order to identify agents. Some authors [13][9] after having discussed roles define the agents assigning one or more roles to them. We still look at the system through the decomposition of its functionalities and therefore we prefer to proceed in the agent identification grouping some functionality into one agent. According to our definition of agent, we can think of an agent as an use case or package of use cases. Therefore, starting from the use-case diagrams of the previous phase we use packages to illustrate the functionalities assigned to each agent. What is important to do is to cover all the use-cases that compose the system functional description.

This is the first step where our approach is strongly different from the others. From the Domain Description diagram, using our Rational Rose Add-in we identify the agent by selecting the option “Identify new agent” from a pop-up menu after having selected all the use-cases that will

compose it. The result is: (a) the creation of the Agent Identification diagram where all the agents are automatically represented as a package and their functionalities described by the use cases selected as part of the agent; (b) the creation of the activity diagram for the task specification of this agent.

C. Role Identification phase

In Fig. 1 we can see that this phase is considered to be part of both the System Requirements and Agent Society models. This is the logical consequence of the fact that here we want to explore scenarios coming from the previous phase. This is a functional/behavioral description of the agent and therefore part of the System Requirement model but it is also a representation of its relations with the other agents and as a consequence a part of the Agent Society model.

We use to illustrate these scenarios with a set of sequence diagrams. This set is identified starting from the analysis of the different paths that is possible to identify in the A.Id. diagram. The roles that the agent can play in its life are not different from the classical object-oriented concept of role of an object. They are represented in the design as objects in R.Id. sequence diagrams using the syntax: <name of the role>:<name of the agent>.

Each communication identified between different agents, needs to be discussed from several different point of views. Referring to FIPA standards [27] we can see the communication as characterized by: a sender and a receiver agent, an interaction protocol, an ontology and a content language. The sender and receiver agents have been identified in this phase, in order to describe the other elements we should refer to the O.D. diagrams of the system.

D. Task Specification phase

In the task specification phase we draw one activity diagram for each agent. We now want to decide which tasks are needed to realize the functionalities described in the previous steps. After having identified the agents and their roles we are now facing the third element of our definition of agent (the task); the last element (the actions) will be described in the phases of the Implementation Model. As already discussed, if we think at a FIPA-based agent implementation of the system each task can be realized with a sub-class of the main agent class and each of the actions within the tasks can be the methods of the task-class.

Each specific T.Sp. diagram is divided into two swimlanes: in the right one we introduce the tasks of the specific agent, in the left one we introduce the tasks of the interacting agents in order to represent the relationships of this agent with the others.

Thanks to the support of our PASSI add-in for Rose, if AgentA sends a message to AgentB and the related task has been specified in the T.Sp. diagram of AgentA, the situation is automatically reported in the diagram of AgentB when we initiate its composition.

E. Ontology Description phase

In the Ontology Description phase we want to describe the

agent society from the ontological point of view. Two elements are therefore particularly important: the domain ontology and the exchange of information among the agents. Although it is possible to discuss both of them in the same diagram we often prefer to create two different diagrams: the Domain Ontology Description (DOD) and the Communication Ontology Description (COD).

In the DOD we represent the ontology as an XML schema [16]. In Fig. 2 we can find an example of our notation. Consider the *IRObstDistance* element. It is mapped to an XSD complextype element composed of the *Direction* attribute (representing the angle between the obstacle position and the robot front direction) and another complextype element, *ObstDist*, that represents the position of the obstacles. The elements presented in this diagram will be used to define the pieces of knowledge of the agents and the ontology of their communications.

The description of the agent's communications is performed using the COD diagram (Fig. 3). It is a class diagram and basically describe the agents' knowledge and the ontology of their communications. Communications according to the FIPA standards are composed of speech acts [17], [18] whose simplest form is: $\langle i, act(j, C) \rangle$ where *i* is the originator of the speech act (we can refer to it as a message), *act* is the name of the act, *j* is the target and *C* the semantic content. In the FIPA Agent Communication Language [19] this can be mapped as follows:

```
(act
:sender i
:receiver j
:content
C)
```

Note that speech acts (act in the example above) are grouped by the FIPA standards in several interaction protocols according to the intention they respond to. This example is however incomplete because it is lacking of the language and ontology specifications outside of which the content *C* makes no sense.

From the previous argumentation we can deduce that for each communication we need to specify three elements: ontology, language and interaction protocol.

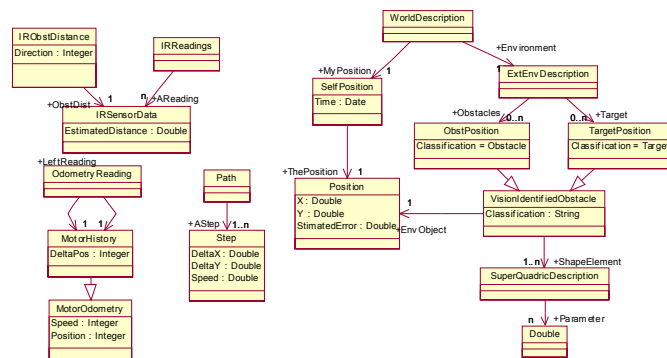


Fig. 2. In the Domain Ontology Diagram the ontology of the system is designed using a syntax that allows an easy generation of XML code for the description of the knowledge.

While several languages and interaction protocols are standardized by FIPA, ontology that is often strictly related to the problem is to be defined in the specific application and therefore we can here refer to the structures defined in the DOD.

For instance, suppose that in a scenario depicted using a R.Id. diagram, the Planner agent asks to the Vision agent information about the environment and the latter replies providing the estimated positions of obstacles, targets and the robot. We can represent this communication in the COD diagram using an association that is drawn from the initiator of the conversation to the other agent. As already discussed each communication is characterized by three attributes. We group them in an association class. This is the characterization of the communication itself (a communication with different ontology/ language/ protocol is certainly different from this one) that is naturally represented as an association because it relates the instances of the two agents class creating a correspondence between the knowledge elements of them.

We can easily deduce the roles played by the agents in the communications, looking at the sequence diagram of the R.Id. phase where these communications has been previously introduced. The contribute by our Rose add-in is that it can automatically identify the roles if there is only one link between the agents in the scenarios of the R.Id. phase or it can suggest the roles if there are several of them.

F. Role Description Phase

In this phase we model the life of the agent looking at its uppermost manifestations: roles. Each agent can play several different roles composing the tasks that constitute its behavioral archive in different ways. In this phase we can introduce the social rules (organizational rules, [14]) and the behavioral rules (as discussed by Newell in his “social level” [15]) coming from the domain where the agent exists. For this purpose, we can use OCL or other form of formal/semi-formal description of the rules according to our preferences and needs.

We represent the Role Description diagram as a class diagram where roles are classes grouped in packages

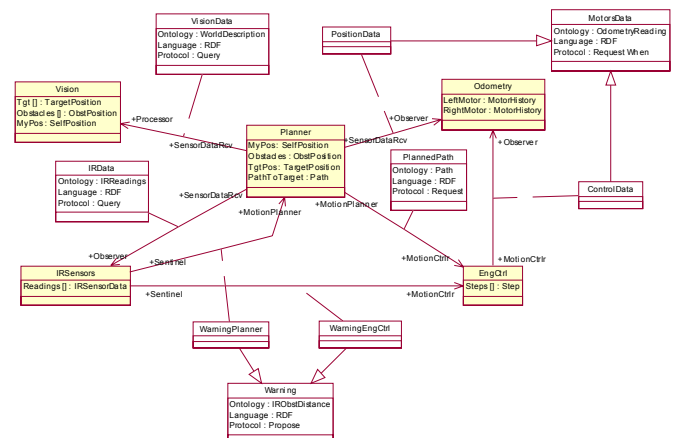


Fig. 3. The Communication Ontology Description (COD) diagram describes the most important aspects of agents' communications and knowledge.

representing the agents. Roles can be connected by relationships representing changes of role, dependencies for a service or the availability of a resource and communications. Each role is obtained composing several tasks and their resulting behavior for this reason we specify the tasks involved in the operation compartment of each class. This clearly expresses (at an abstraction level that is obviously still high) the abilities involved in the role and it can be helpful in the identification of reusable patterns.

As already discussed, in the Role Identification phase we produce a series of sequence diagrams where several roles are identified. Several activities are automatically performed by our Rose add-in for the construction of the RD diagram: (1) It begins the creation of the Role Description diagrams collecting these roles and introducing them as classes in the diagram. Each agent is represented by a package and its roles are disposed in it. (2) If two roles are present in the same sequence diagram and are connected by a link that establishes a first-second relationship between them, the PASSI add-in connects them in the R.D. diagram with a Role Change relationship. (3) If two agents are connected in the Communication Ontology Description diagram then a communication relationship (represented with a solid line) is introduced in this diagram between the corresponding roles using the role labels of the relation in the COD diagram as the keys to identify them.

As already discussed we introduce in this diagram rules about the behavior of the agent. In Fig. 4 we have detailed the trigger conditions for the change of role using notes attached to the relationships. For example the Vision agent changes from the Grabber to the Processor role each time a new image

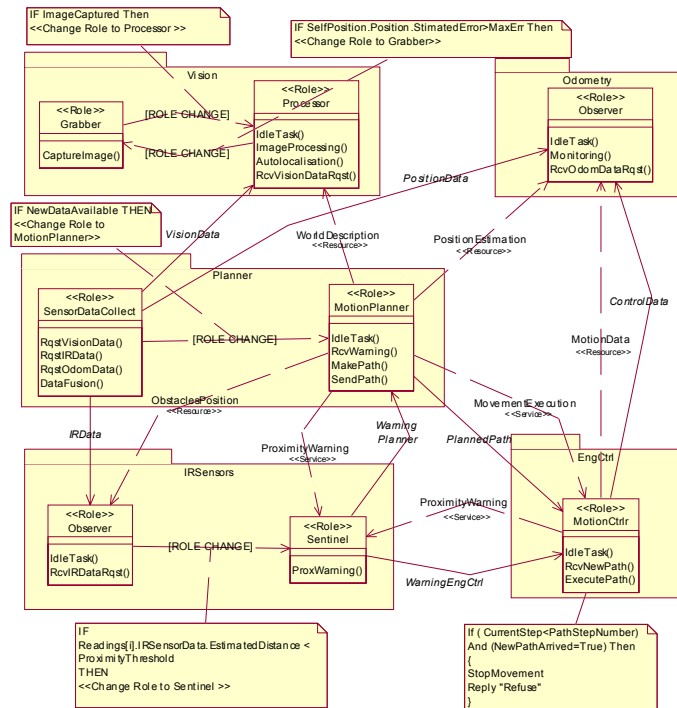


Fig. 4. The Role Description (RD) diagram illustrates the change of roles, the communications and the dependencies among the roles.

is captured and ready to be analyzed. We represent the change of role as a dependency relationship because we want to represent the dependency of the second role from the first for the execution of some actions or the realization of a condition. Sometime the trigger condition is not explicitly generated the first role but its precedent appearance in the scenario justifies the consideration that it is necessary to prepare the situation that allows the second role to start.

G. Protocols Description phase

Several interaction protocols can be found in the FIPA standards but it is possible that the designer needs a new specific pattern of interaction and therefore decides to create a new one. If this is the case he can describe the new protocol in form of AUML sequence diagrams as discussed in [20]. In this way he will define the initial communicative act, the possible replies and the steps that will conclude the communication.

H. Agents Structure Definition and Agents Behavior Description phases

The design of the structure of MAS can be divided into two logical levels of abstraction. In the first we look at the agent society level (we call it multi-agent level) and each agent is regarded as an element of this society. In the second, more detailed, level (called single-agent level) we consider only one agent at a time and we look at its fully detailed structure with a granularity that can be used to produce code.

The Multi-Agent Structure Definition is represented with a class diagram (see Fig. 5); each agent is represented by the main agent class whose elements are: (i) the attributes representing the knowledge as already discussed in the ontology description, (ii) the methods representing the tasks already identified in T.Sp. phase. The relationships connecting two agents represent the communications existing between them and this is the natural extension of the object-oriented concept of association to the agent world. While an object can instantiate another class to obtain a service from it, this is not allowed to an agent. It cannot force another agent to do something, it can only ask for the service (with a communication) that will be provided only if the other agent agrees.

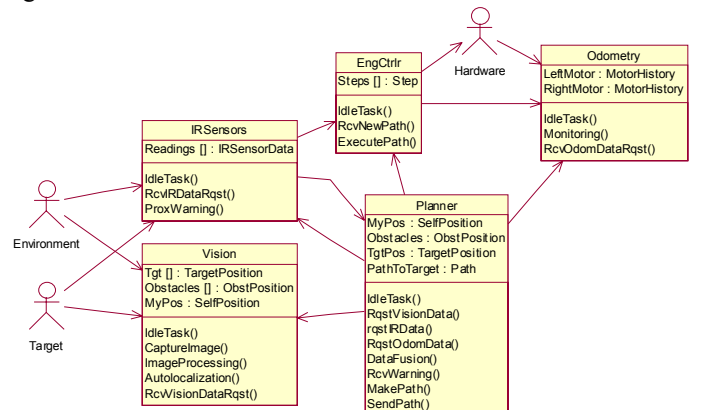


Fig. 5: The Multi-Agent Structure Definition (MASD) diagram is a class diagram representing the structure of the system with a compact notation.

The elements of this diagram are already present in the previous phases of the design and therefore it is automatically drawn by the PASSI add-in. The names of the agents come from the A.Id. phase, their knowledge from the communication ontology description, their tasks and communications (represented as association in this diagram) from the T.Sp. and R.D. diagrams.

The Single-Agent Structure Definition phase is composed of one different class diagram for each agent. Here we fully exploit the structure of the agent detailing the main agent class (inherited from the base agent class) with its attributes and methods (for example constructor, destructor and the methods needed to register the agent in the white/yellow pages directories); each task of the agent is represented as a class (inherited from the base task class) with the attributes needed to perform the specific series of activities and the related methods.

In the Multi-Agent Behavior Description we use an activity diagram (see Fig.6) to represent the behavior of the system at a level of detail that arrives to consider the single method of the each agents/tasks. A swimlane is introduced for each class (representing the implementation of an agent or a task), in this place we introduce activities that represent methods of the class. Each activity is connected with the others to represent the flow of control and the events of the system. In our experience this diagram represents the most useful level of detail for the representation of a MAS. Several times we obtained very large diagrams from this phase but they always proved significant both for the exploration of the behavior and for the debug of the system.

In the Single-Agent Behavior Description phase we focus on the implementation of the methods of the classes. We can design them using classical approach (like flowcharts, state diagrams and so on) or even semi-formal text descriptions.

I. Code Production Phase

Many CASE tools offer the possibility of automatically generate the skeletons of the classes from their class diagrams and some efforts can be done also in the production of the inner parts of the methods. In order to obtain better results, we are following a slight different approach; thanks to the structure of the FIPA architecture implementation we have successfully identified a series of standard pieces of code that can be introduced in the body of the methods when some graphical descriptions are found in the different steps of the design.

We are also moving forward preparing the extraction from the design diagrams of a representation of the agent in a meta-language (XML-based) that we plan to use in order produce the code for different FIPA platform implementations.

J. Deployment Configuration

This phase is particularly important if we deal with mobile agents and with significant problems in the dissemination of the agents of the system. The deployment configuration diagram describes where the agents are located and which different elaborating units need to communicate in order to permit the communications among the agents. As usual, elaborating units are shown as 3-D boxes. We show agents as components, communications among agents are represented by dashed lines with the communicate stereotype, directed as in the R.D. diagram. For each communication described in the R.D. diagram occurring between agents in different elaborating units, a dashed line is drawn. The receiving agent has an interface to show that it is capable of dealing with that communication. (i.e. it understands the protocol used.) We also use an extension of the UML syntax is used in order to deal with mobile agents moving from one computer to another. It is represented by a dashed line with the move_to stereotype.

IV. CONCLUSIONS

We discussed PASSI, a design methodology for multi-agent systems, and the support that it can receive by a specific add-in we produced for Rational Rose. The use of UML with minor extensions and the focus on highly structured implementation platforms like the FIPA-compliant ones gave us the opportunity of providing the designer with a very helpful support both in the design activity and the code production phase.

The different steps of PASSI are strictly connected in order to permit the representation of MAS taking into account different levels of abstractions and different points of view. Important concepts of MAS like communications, ontology and roles are present in more than one moment of the process related to different aspects of the design. Ontology for example is defined in the Domain Ontology Description, used in the messages of the Communication Ontology Description and it is part of the code production phase since we can extract XML code from the diagram and we can use it as a content language for the messages composing the communication.

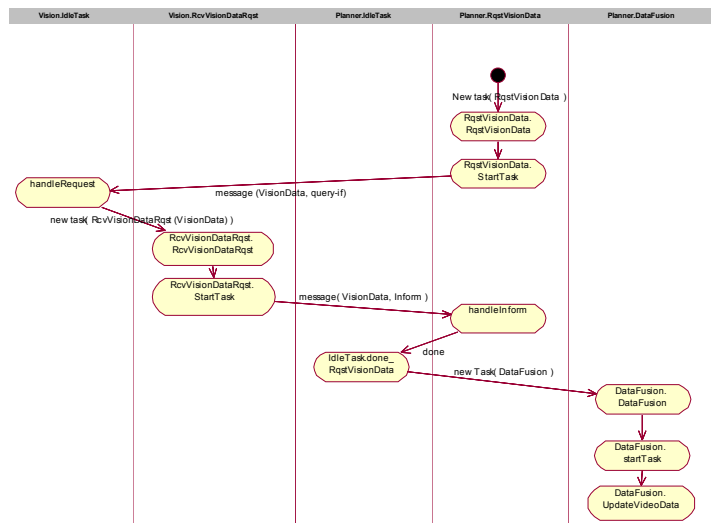


Fig.6. The Multi-Agent Behavior Description (MABD) represents the behavior of all the agents using swimlanes to separate the responsibilities of each class (agent/task).

We are still working to enhance the support offered by our tool. Using a connection with a commercial word-processor we are going to automatically produce the documentation of the design using specific templates prepared for agent-based systems. Great efforts can still be done in the code generation introducing a library of patterns of agents/tasks and the capability of automatically composing both the design and the code starting from the functionalities of the agent/task selected from the library. We are aware of the difficulty of the problem but this is drastically simplified by the boundaries of our choices: a fixed implementation platform (FIPA-OS/JADE,...) allows us to prepare largely reusable parts of the system elements, the use of XML for ontology description allows us to skip (or largely reduce) the manual work of the designer in the creation of the agent knowledge, the deduction of an XML representation of the agent code from the design allows a simple porting to the different platforms.

REFERENCES

- [1] Chella, A., Cossentino, M., and Lo Faso, U. Applying UML use case diagrams to agents representation. Proc. of AI*IA 2000 Conference. (Milan, Italy, Sept. 2000).
- [2] Chella, A., Cossentino, M., and Lo Faso, U. Designing agent-based systems with UML in Proc. of ISRA'2000 (Monterrey, Mexico, Nov. 2000).
- [3] Chella, A., Cossentino, M., Infantino, I., and Pirrone, R. An agent based design process for cognitive architectures in robotics in proc. of WOA'01 (Modena, Italy, Sept. 2001).
- [4] Chella, A., Cossentino, M., Infantino, I., and Pirrone, R. A vision agent in a distributed architecture for mobile robotics in Proc. Of Worskshop "Intelligenza Artificiale, Visione e Pattern Recognition" in the VII Conf. Of AI*IA (Bari, Italy, Sept. 2001).
- [5] Chella, A., Cossentino, M., Tomasino, G. An environment description language for multirobot simulations in proc. of ISR 2001 (Seoul, Korea, Apr. 2001)
- [6] Antón, A.I., McCracken, W.M., and Potts, C. Goal Decomposition and Scenario Analysis in Business Process Reengineering in proc. of Advanced Information System Engineering: 6th International Conference, CAiSE '94 (Utrecht, The Netherlands, June 1994) 94-104.
- [7] Antón, A.I., and Potts, C. The Use of Goals to Surface Requirements for Evolving Systems, in proc. of International Conference on Software Engineering (ICSE '98), (Kyoto, Japan, April 1998), 157-166
- [8] van Lamsweerde, A., Darimont, R. and Massonet, P. Goal-Directed Elaboration of Requirements for a Meeting Scheduler: Problems and Lessons Learnt in Proc. 2nd International Symposium on Requirements Engineering (RE'95) (York, UK, March 1995), 194-203
- [9] DeLoach, S.A., Wood, M.F., and Sparkman, C.H. Multiagent Systems Engineering. International Journal on Software Engineering and Knowledge Engineering 11, 3, 231-258.
- [10] Jacobson, I., Booch, G., Rumbaugh, J. The Unified Process. IEEE Software (May/June 1999).
- [11] Jacobson, I., Christerson, M., Jonsson, P., and Overgaard, G. Object-Oriented Software Engineering: A Use Case Driven Approach. Addison-Wesley (1992).
- [12] Potts, C. ScenIC: A Strategy for Inquiry-Driven Requirements Determination in proc. of IEEE Fourth International Symposium on Requirements Engineering (RE'99), (Limerick, Ireland, June 1999), 58-65.
- [13] Wooldridge, M., Jennings, N.R., and Kinny, D. The Gaia Methodology for Agent-Oriented Analysis and Design. Journal of Autonomous Agents and Multi-Agent Systems. 3,3 (2000), 285-312.
- [14] F. Zambonelli, N. Jennings, M. Wooldridge. Organizational Rules as an Abstraction for the Analysis and Design of Multi-agent Systems. Journal of Knowledge and Software Engineering, 2001, 11, 3, 303-328.
- [15] Newell, A. The knowledge level, Artificial Intelligence, 18 (1982) 87-127.
- [16] David Carlson. Modeling XML Applications with UML: Practical E-Business Applications. Boston: Addison-Wesley, 2001.
- [17] Searle, J.R., Speech Acts. Cambridge University Press, 1969.
- [18] FIPA Communicative Act Library Specification. Foundation for Intelligent Physical Agents, Document FIPA00037 (2000). <http://www.fipa.org/specs/fipa00037/>.
- [19] FIPA ACL Message Structure Specification. Foundation for Intelligent Physical Agents, Document FIPA XC00061E. <http://www.fipa.org/specs/fipa00061/XC00061E.html>.
- [20] J.Odell, H. Van Dyke Parunak, B. Bauer. Representing Agent Interaction Protocols in UML, Agent-Oriented Software Engineering, P. Ciancarini and M. Wooldridge eds., Springer-Verlag, Berlin (2001), 121-140.
- [21] Jennings, N.R. On agent-based software engineering. In Artificial Intelligence, 117 (2000), 277-296.
- [22] DeLoach, S.A., Wood, M.F., and Sparkman, C.H. Multiagent Systems Engineering. International Journal on Software Engineering and Knowledge Engineering 11, 3, 231-258.
- [23] Aridor, Y., and Lange, D. B. Agent Design Patterns: Elements of Agent Application Design. In Proc. of the Second International Conference on Autonomous Agents (Minneapolis, May 1998), 108-115.
- [24] Kendall, E. A., Krishna, P. V. M., Pathak C. V. and Suresh C. B. Patterns of intelligent and mobile agents. In Proc. of the Second International Conference on Autonomous Agents, (Minneapolis, May 1998), 92-99.
- [25] F. Zambonelli, N. Jennings, M. Wooldridge. Organizational Rules as an Abstraction for the Analysis and Design of Multi-agent Systems. Journal of Knowledge and Software Engineering, 2001, 11, 3, 303-328.
- [26] Wooldridge, M., Jennings, N.R., and Kinny, D. The Gaia Methodology for Agent-Oriented Analysis and Design. Journal of Autonomous Agents and Multi-Agent Systems. 3,3 (2000), 285-312.
- [27] FIPA Abstract Architecture Specification (Refinements). Foundation for Intelligent Physical Agents, Document FIPA PC00094 (2001). <http://www.fipa.org/specs/fipa00094/PC00094.html>
- [28] Collinot A., Drogoul A. Using the Cassiopeia Method to Design a Soccer Robot Team. Applied Artificial Intelligence (AAI) Journal, 12, 2-3 (1998), 127-147.