Tools and patterns in designing multi-agent systems with PASSI

ANTONIO CHELLA[1,2], MASSIMO COSSENTINO[1], LUCA SABATUCCI[1]

[1]ICAR/CNR – Istituto di Calcolo e Reti ad Alte Prestazioni/Consiglio Nazionale delle Ricerche c/o CUC, Viale delle Scienze, 90128 Palermo, ITALY

[2]DINFO - Dipartimento di Ingegneria Informatica, Università degli Studi di Palermo
Viale delle Scienze, 90128 Palermo, ITALY

Abstract: In the last years the increasing attention on multi-agent systems (MAS) emphasized the need of a quality software engineering approach to their design and realization. In this paper we propose a comprehensive approach for the development of MAS oriented applications that uses a complete design methodology addressing important issues such as ontology representation, agent collaboration and patterns reuse. The PASSI (Process for Agent Societies Specification and Implementation) design methodology is supported by a specific design tool, granting a great number of automatisms during the design, and a pattern repository for the reuse practice; these are determinant in cutting down the time and cost for developing these systems. During the description of PASSI and its supporting tools, we refer to the most diffused standard for agents (FIPA - Foundation for Intelligent Physical Agents) and modeling languages (UML and AUML). We will complete our discussion with some examples of the functionalities offered by these tools.

1 Introduction

Several proposals exist for designing and representing agent-based systems [3][4] [5][6]. Because research about agency is still an open topic, some proposed representations involve abstractions of social phenomena and knowledge [3][5]. Others address implementation issues and have higher fidelity models (e.g. [4][6]).
One response to these developments is to treat agent-based software the same as other types. That is, there is no need for methods or representations specifically for agent-based software. We reject this view because it is more natural to describe agents in a psychological and social language.
As a consequence, we think that design methodologies and even design tools that we commonly use to design object-oriented systems are not the best way to work on the agent-oriented software. Many aspects of the object-oriented world can be profitably reused but a proper support of the agents peculiarities is necessary and this should originate new methodologies and specifically conceived design tools.
We propose PASSI [8] (Process for Agent Societies Specification and Implementation) and the tools that support it (*PTK* and *AgentFactory*) as a solution to the above arguments. This methodology is the result of a long period of study and experimentation mainly in robotics [7][9]. It is composed of five models (System Requirements, Agent Society, Agent Implementation, Code Model and Deployment Model) which include several distinct phases. The advantages of using PASSI rather than other methodologies, derive from the complete support that is has from *PTK* and *AgentFactory*. *PTK* is an add-in for a diffused commercial UML-based CASE tool (Rational Rose) that can enhance the design robustness and coherence also lowering the designer effort. It provides a strong support for the code production phase with the automatic generation of a great amount of code. *AgentFactory* is a pattern reuse application that allows the rapid prototyping of great parts of complex FIPA compliant systems [10] with a few mouse clicks.
The following sections are organized as follows: in section two we discuss, with some examples, the support that the *PTK* tool provides to the designer during his/hers work with PASSI. In section three we illustrate our approach to patterns reuse and how it is supported by the *AgentFactory* application. Finally, in section four some conclusions are drawn.

2 Designing agents with a specific CASE tool

In the following sections, we will provide a short description of the contribution offered by the *PTK* (PASSI Toolkit) tool in designing a FIPA compliant system using the PASSI [8] methodology. PASSI is composed of five models (see Figure 1) that address different design concerns and twelve steps in the process of building a model. In PASSI we refer to the most common standards in software engineering and agents (UML, AUML, FIPA, XML, RDF). PASSI is the unique methodology that offers a specific support for robotics with a consistent number of patterns already present in our repository together with a

complete design process from requirements elicitation to coding and deployment.

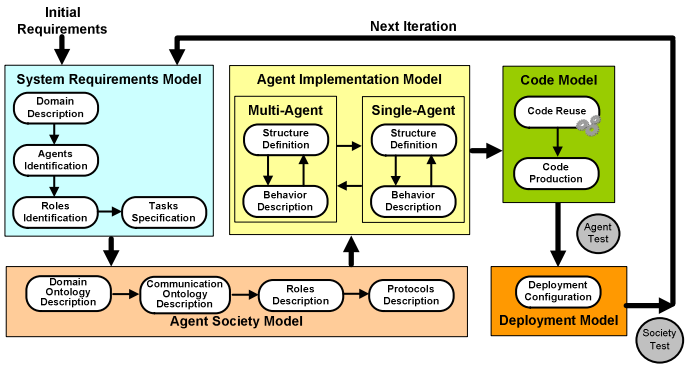

Figure 1. The models and phases of the PASSI methodology

Our work about *PTK*, starts from the consideration that most commercial CASE tools are only object-oriented. We believe that the support of an agent-oriented CASE tool can simplify the MAS designer's work, increase the reuse of code (through a database of agents/tasks patterns), and permit the automatic production of a considerable part of the code. Moreover, our tool helps untrained users to follow a proper software engineering approach.

The philosophy underling the work with *PTK* is that designing a MAS corresponds to instantiate a meta-model of the multi-agent system that fulfills the requirements of the specific problem. For this reason, specific choices (for example the designation of the agents and their functionalities) have a direct consequence on the other steps of the process. The result of this approach is that the different diagrams that constitute the final result of designing with the PASSI methodology are gradually composed by the author and the *PTK* tool in their interaction. Some of these diagrams are totally dependent by the designer, some are automatically built by the tool and others are partially composed by the tool and then completed by the designer. The add-in can also produce a report of the entire design in a Microsoft Word format. Together with the diagrams, the document will contain textual descriptions and some tables summarizing the agents, their behaviors, roles, communication, ontology, etc.

In the following we will report some examples that briefly illustrate the main features of *PTK* (the PASSI ToolKit). We do not intend to provide an unique complete design case study but just some scenarios describing how the designer can take advantage from the use of a design methodology and a CASE tool specifically conceived to support it.

The discussion is therefore articulated into a series of subsections dealing with the main PASSI phases and describing in each of them, both the designer work and the tool contribute.

2.1 Domain Description and Agent Identification phases

We describe requirements in terms of use case diagrams, and as a result, the Domain Description phase is a functional description of the system composed of a hierarchical series of use case diagrams. Stereotypes used here come from the UML standard. Suppose that our system is very simple and it is represented by the three use cases of Figure 2.

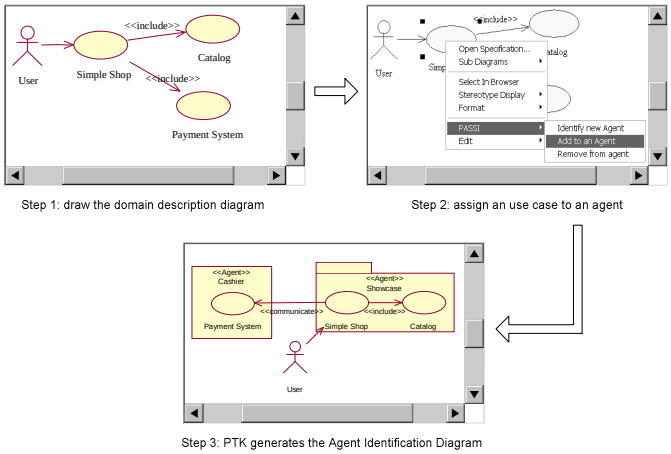

Figure 2. Three different phases of a toy system design. Up on the left the Domain Description with three use cases, on the right the agent identification phase with the PTK context menu and finally down the resulting A.Id. diagram

Looking at the specific functionalities we can decide how to distribute them into several different agents (Agent Identification phase). For example we can decide to group use cases *SimpleShop* and *Catalog* in the same agent (*Showcase*) and implement the functionality described by use case *PaymentSystem* in another agent (C*ashier*). Each agent will be responsible to provide the system with the functionalities described in its use cases.

PTK helps us in performing this agent identification operation using a context menu (see Figure 2). In the resulting diagram (Agent Identification), the two identified agents are represented as packages containing the assigned use cases. This diagram has been automatically composed by *PTK* using the information provided in the previous steps.

2.2 Roles Identification phase

This phase occurs early in the requirements analysis since we now concern more with an agent's externally visible behavior rather than its structure – only approximate at this step. Roles identification is based on exploring scenarios arising from the Agents Identification diagram. Scenarios are depicted using UML sequence diagrams where each object represents a role; an agent may participate in different scenarios playing distinct roles in each of them. It may also play distinct roles in the same

scenario. Roles described in this phase contribute to build the model of the system and the tool being aware of them (and of their belonging to some agents) will use this information to build part of the following diagrams (for example these roles will be reported in the Roles Description diagram).

2.3 Task Specification phase

At this step, for each agent we focus on its behavior in order to decompose it into simpler tasks. Tasks generally encapsulate some functionality that forms a logical unit of work.

A Task Specification diagram summarizes what the agent can do, ignoring information about roles that an agent plays when carrying out particular tasks.

For every agent in the model, we draw an UML activity diagram that is made up of two swim-lanes. The one from the right-hand side contains a collection of activities symbolizing the agent's tasks, whereas the left-hand side one, contains some activities representing the other interacting agents.

The support provided by *PTK* in working with these diagrams consists in automatically synchronizing the diagrams of different agents; in fact, if describing agent *A* the designer introduces a communication involving the behavior *X* of the agent *B*, this behavior will be automatically reported to the agent *B* Task Specification diagram. This ensures an high level of consistency among the different diagrams and representations of agents.

2.4 Ontology Description phase

An agent-based system may achieve its semantic richness through explicit ontologies, or domain-specific terminologies and theories. In order to detail the resulting ontology of the solution we introduce the O.D. (Ontology Description) phase that is composed of two sub-phases: in the first, the D.O.D. (Domain Ontology Description) diagram, we describe the ontology of the domain representing the involved entities through classes; in the second, the C.O.D. (Communication Ontology Description) diagram, the focus is on the agents' knowledge and the communicative relationships among these agents.

2.4.1 The Domain Ontology Description

Our domain ontology is described in terms of concepts (entities of the domain), predicates (stating the value of properties of domain entities) and actions (that can be operated in the domain). We represent these elements and their relationships using an UML class diagram (called D.O.D., Domain Ontology Description, in the PASSI methodology). In Figure 3 we can see part of the ontology of a robotics surveillance application. The *GenericComponent* concept (concepts have a yellow fill-color) in this figure represents an element of the environment that the robot can see during its exploration. About an instance of this concept (for example the *GenericComponent* identified by the ID = 12) we could express a proposition (*IsIntruder* predicate, blue filled) stating if it is an intruder or not. We can similarly introduce actions and relate them to the concepts they will affect (for example the *Localize* action regards the act of localizing the position of a specific *GenericComponent* in the environment).

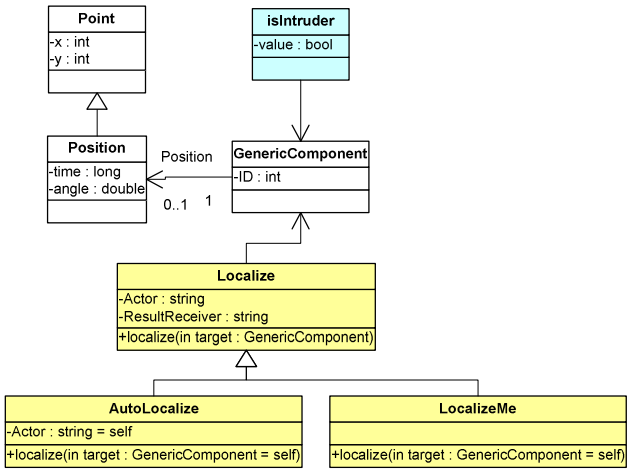

Figure 3. An example of Domain Ontology Diagram (DOD) representing part of the ontology of a robotics application

Once the ontology design is completed, *PTK* offers a wizard to generate the corresponding RDF, and it is also possible to export the diagram in the XMI format.

2.4.2 The Communication Ontology Description

The Communication Ontology Description (COD) phase is the next PASSI step. In the COD we describe the agents' knowledge and their communications with a class diagram.

Each FIPA communication is characterized by three elements: (1) the ontology (portion of knowledge exchanged); (2) the agent interaction protocol (that designates the sequence of communicative acts); and the content language (the language used to code the message content).

PTK offers a great support in composing diagrams like these. Agents and (some) communications are automatically introduced by the tool starting from the results of the R.Id. (sequence diagrams describing scenarios) and T.Sp. (activity diagrams describing the behavior of each agent) phases. A class is introduced for each identified agent, and a relationship is drawn among the agents involved in the exchange. The designer can complete the specification of each communication using the form described in Figure 4. There he/she can select:

- a name for the communication;

- an interaction protocol from the list of communication patterns included in the repository;
- the ontology of the communication from the elements defined in the DOD;
- the content language (if he/she adopts RDF then the automatic generation of a lot of code will be available);
- the task that in each agent will deal with this communication (the list of tasks comes from the tasks defined in the T.Sp. phase).

As a consequence of the ontology selected by the designer, it will be possible for *PTK* to define the proper data structure that in each agent can store it.

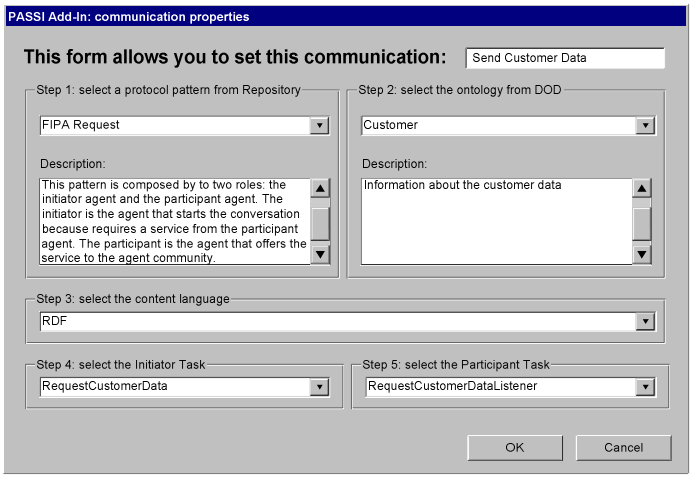

Figure 4. This form is used to set the communication properties (interaction protocol, ontology, content language and involved tasks)

When the Communication Ontology Description phase is completed, the designer can export the RDF code of each message and moreover, *PTK* will generate the necessary code (specific for the FIPA agent platform used) to deal with the communication (this also includes the conversion JAVA data structure/RDF/JAVA data structure, of information exchanged by agents).

2.5 Roles Description and Protocols Definition phases

The Roles Description (RD) phase models the lifecycle of an agent taking into account its roles, the collaborations it needs and the conversations it is involved in. The RD phase yields a collection of class diagrams in which classes are used to represent roles. Each agent is symbolized by a package containing roles' classes. Role are obtained composing several tasks in the resulting complex behavior. In this diagram we also report change of roles, communications and dependencies among agents. Many of these elements are again automatically introduced by *PTK* and usually the designer has only to introduce a few of them.

Commonly, we only use standard FIPA interaction protocols. In this case the designer does not need to perform the Protocols Definition phase (done with AUML if necessary).

2.6 Agents Structure and Behavior Definition

The Agent Structure Definition phase produces several class diagrams logically subdivided into two views: the multi-agents and the single-agent view. In the former, we call attention to the general architecture of the system and so we can find agents and their tasks in it. In the latter, we focus on each agent's internal structure, revealing all the attributes and methods of the agent class together with its inner tasks' classes.

It is interesting to note that both of these diagrams are automatically built by the *PTK* using the information coming from the previous steps (and some knowledge about the class structure and organization of the agents that are typically different among the FIPA platform implementations); this ensures an high level of coherence between the implementation level of abstraction and the previous stages of the design.

The next phase is the Agent Behavior Description that produces several diagrams subdivided into the multi-agent and the single-agent views. In the former, we draw the flow of events by methods invocation and the messages exchange. In the latter, we feature the agent behaviors' methods.

2.7 Code Generation and Reuse

The *PTK* add-in can generate the code for all the skeletons of the agents, tasks and other classes included in the project. It does not use the standard Rational Rose functionality since we also want to allow the reuse of patterns coming from our repository.

The pattern repository consists of a series of reusable portions of agents and tasks. For example the designer can take from this archive a *generic agent* (that has the capability of registering itself to the basic platform services) and he/she can introduce it in the actual project.

The repository also includes a list of tasks that can be applied to existing agents. For example we have tasks dedicated to deal with the initiator/participant roles in the most common communications. When a pattern is introduced in the design, not only some diagrams (like the structural and behavioral one of the implementation level) are updated but the resulting code also contains large portions of inner parts of methods; the result is an highly affordable and quick development production process. More functionalities are provided by the *AgentFactory* application (see next section) that will be integrated with the future releases of *PTK*.

3 The use of patterns in the design

Many researchers have proposed to expand the traditional pattern model towards the agent paradigm [1][2][9].
In our approach, we aim to largely apply a properly defined concept of agent pattern and we built an application (*AgentFactory*) that supports it. *AgentFactory* can, very quickly, create complex multi-agent systems using a large repository and can also provide the design documentation of the composed agents. The tool can work online as a web-based application[1], but can also be used as a stand alone application.
Our patterns result from the composition of three different aspects of a multi-agent system:

1) the static structure of one or more agent(s) or parts of them (i.e. behaviors),
2) the description of the dynamic behavior expressed by the previous cited elements
3) the programming code that realizes both the static structure (skeletons) and the dynamic behavior (inner parts of methods) in a specific agent platform context.

In the following sub-section we will report a pattern of agent and we will use it to demonstrate the functionalities of our *AgentFactory* tool.

3.1 An example of pattern: the Explorer agent

The pattern we are going to analyze is a mobility pattern that allows the exploration of a remote platform with the intent to search for some information. A typical scenario that illustrates the aim of this pattern is represented by web searching, where an agent has to recover some data from a remote platform; using this pattern, the agent does not move itself into the remote platform but it delegates this work to an *explorer* agent. The pattern (Figure 5) includes two collaborating agents: the *base* and the *explorer* agent. The *base* agent has the ability of creating one or more *explorer* agents, giving to each of them the address of a remote platform. The *explorer* agent can move to the target platform and eventually perform some kind of searching (operation not provided by this pattern). When the explorer has found the information in the remote platform it forwards the data to the base agent and then terminates itself.

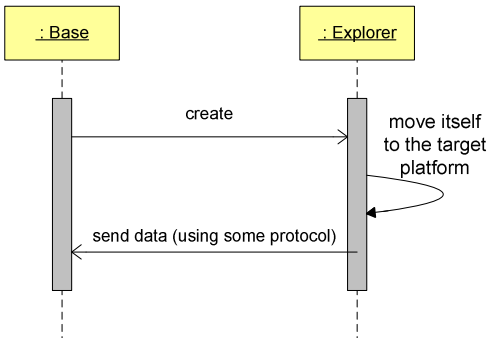

Figure 5. The explorer pattern consists in a *base* agent who delegates to another (called *explorer*) the task of moving to another platform in order to search for some kind of information

Both the *explorer* agent and the *base* agent require the ability to communicate according to a specific protocol, in order to exchange the collected data. Because the protocol is not specified in this pattern, this feature should be obtained by the application of a communication pattern.

3.2 The *AgentFactory* Tool

The *AgentFactory* tool allows the automatic generation of the pattern code for both the JADE and FIPA-OS that are among the most diffused FIPA compliant agent platforms. This has been obtained using a meta-pattern language description of the elements of our pattern repository.
All of the features of each agent/tasks pattern (that can be different from one platform to the other), are named with a meta-label. For example the agent super-class that is named "*AgentShell*", becomes "*Agent*" in the JADE code instantiation and "*FIPAAgent*" in the FIPA-OS one. When the tool generates the code, it applies a pre-transformation in which all the meta-labels are substituted by the correct names.
As we know, the application of a pattern to an existing system causes a significant change in its structure, and this often implies other modifications in order to join the new elements with the existing ones. This is another of the features offered by our tool: patterns are completed by a collection of *constraints* that describe how the target system has to be changed in order to correctly accept the new parts.

[1] Available at: http://mozart.csai.unipa.it/af/

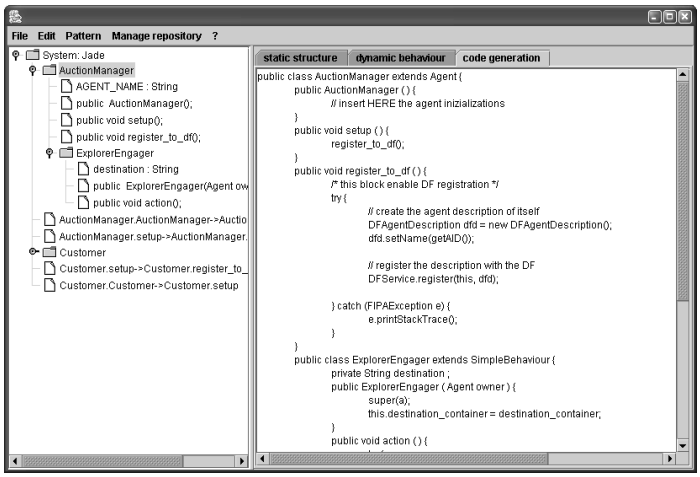

Figure 6. The *AgentFactory* tool during the composition of the code for the Explorer pattern

In Figure 6, we can see the *AgentFactory* tool during the code generation phase for the *Explorer* pattern. We completed the system using the *FIPARequest* pattern in order to introduce the communication capability in both the agents.

What should to be considered is that with a few mouse clicks and selecting the *Explorer* and *FIPARequest* patterns, in a few seconds we can produce an application composed of two agents, six classes and about 190 lines of code (including comments). The documentation of that is provided by an UML class diagram (structure of the system) and UML activity diagram (behavior of the system).

4 Conclusions

We discussed PASSI, a design methodology for multi-agent systems, and the support that it can receive by a specific add-in (*PTK*, PASSI ToolKit) we produced for Rational Rose and a repository of patterns supported by the *AgentFactory* application. The use of UML with minor extensions and the focus on highly structured implementation platforms like JADE or FIPA-OS gave us the opportunity of providing the designer with a very helpful support both in the design activity and the code production phase.

PASSI explicitly pursues the following goals: (i) a great attention for standards (we used UML, FIPA, XML, RDF); (ii) providing a complete design process from requirements elicitation to implementation and deployment; (iii) a specialization for the design of robotic applications with the availability of specific robotic patterns.

The *PTK* tool guides the designer throughout all the design work, allowing the automatic compilation of several diagrams and the generation of the code for the agent skeletons. The generation of the remaining part of the code (inner parts of methods) is delegated to a repository including several patterns of agents and behaviors. Using a connection with a commercial word-processor (Microsoft Word) the tool automatically produces the documentation of the design adopting specific templates prepared for agent-based systems.

Although this tool offers interesting results our research goes beyond it and we produced the *AgentFactory* application that refers to a more complex and complete concept of pattern. The pattern is seen in the most common significance of design pattern but adapted to the specific demands of agents. The use of several levels of meta-representations of these patterns allows the generation of complete agents or of parts of them (also including the behavioral part of the code) for the FIPA compliant platforms as Jade or FIPA-OS.

We have completed several projects using the PASSI methodology and our tools particularly in robotics and information systems. Experimental results have been very encouraging and we are now working in the direction of extending the number of patterns in order to maximize the possible applications of their repository.

References

[1] 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.

[2] 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.

[3] Bernon, C., Gleizes, M.P., Peyruqueou, S., and Picard, G., Adelfe, a methodology for Adaptive Multi-Agent Systems Engineering. Third International Workshop "Engineering Societies in the Agents World" (ESAW-2002), 16-17 September 2002, Madrid.

[4] Giunchiglia, F., Mylopoulos, J., and Perini, A. The tropos software development methodology: processes, models and diagrams. The First International Joint Conference on Autonomous Agents & Multiagent Systems, AAMAS 2002, July 15-19, 2002, Bologna, (Italy)

[5] 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.

[6] 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.

[7] Chella, A., Cossentino, M., Pirrone, R., Ruisi, A.: Modeling Ontologies for Robotic Environments. Proc. of the Fourteenth International Conference on Software Engineering and Knowledge Engineering. Ischia, Italy, July 2002

[8] Cossentino M., and L. Sabatucci L., Agent System Implementation in “Agent-Based Manufacturing and Control Systems: New Agile Manufacturing Solutions for Achieving Peak Performance”. M. Paolucci, R. Sacile Editors. CRC Press. ISBN: 1574443364. April 2004.

[9] Cossentino, M., Sabatucci, L., Chella, A.: A Possible Approach to the Development of Robotic Multi-Agent Systems. IEEE/WIC Conf. on Intelligent Agent Technology (IAT'03). Halifax (Canada). October, 2003.

[10] FIPA – Foundation for Intelligent and Physical Agents – www.fipa.org