The PASSI and Agile PASSI MAS meta-models

Antonio Chella[1],[2], Massimo Cossentino[2], Luca Sabatucci[1], and Valeria Seidita[1]

[1] Dipartimento di Ingegneria Informatica (DINFO)
University of Palermo
Viale delle Scienze, 90128 -Palermo- Italy
[2] Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR)
Consiglio Nazionale delle Ricerche(CNR)
Viale delle Scienze, 90128 -Palermo- Italy
chella@unipa.it, cossentino@pa.icar.cnr.it, sabatucci@csai.unipa.it,
seidita@csai.unipa.it

Abstract. A great number of processes for multi-agent systems have been presented in last years to support the different approaches to agent-oriented design; every process is specific for a particular class of problems and it deals with a specific MAS meta-model. These differences produce inconsistences and overlaps: a MAS meta-model may define a term not referred by another, or the same term can be used with a different meaning.
We think that the lack of a standardization may cause a significant delay to the diffusion of the agent paradigm outside research contexts, and working for this unification goal, it is also necessary to unambiguously define the terms of the agent model and their relationships thus obtaining a unified MAS meta-model. In this work we propose the PASSI MAS meta-model and the results of its adaptation to the needs of an agile process (agile PASSI).

1 Introduction

Overall the years it has been accepted that agents, rather than just a technology, represent a new paradigm that proved particularly suitable to approach complex software systems.
In order to approach the design and the implementation of a multi-agent system (MAS) in a rigorous way, many modeling methods have been explored: all of these processes, during the development, address high level terms such as agent, goal, role, task and collaboration: they represent the key issues of this type of approach to the problem. The description of the elements involved in a design process, and their relationships, represent one of the fundamental steps of defining a new design process: its MAS meta model (MMM) definition. Hence, the design of a system may be seen as the instantiation of the correspondent MAS meta-model in order to fulfill some specific problem requirements.

The various agent oriented design processes, presented in these years, are significantly different: goal-oriented, situation-oriented, requirement-oriented are examples of different philosophical approaches represented in some of the existing agent oriented processes [1][2][3][4][5]. Each approach suggests a different way to face modeling; this because the system is observed from different perspectives, with the aim of modeling different aspects, considering a different theoretical background or a specific application context. Besides each process forces the designer to assign an implicit meaning to each MMM component and this is often not coherent with the choices of other authors.
Even if this variety of methods may be viewed as a richness, the differences among their meta-model components could create some perplexities when a designer moves from a design process to another, or when two designers try to communicate about a shared solution.

The purpose of this work is to present a description of the MAS meta-models of two design processes that use a quite different approach to face the problem: PASSI[3, 6] and Agile PASSI[7, 8]. PASSI is a requirement driven agent-oriented iterative-incremental design process inspired by a multi-perspective approach that requires a large number of models and views, while Agile PASSI is a methodology derived from PASSI, defined to meet the manifesto for Agile Software Development prescriptions [9].
This paper is structured as follows: in section 2 we present the PASSI and Agile PASSI design processes while their meta-models are described in sections 3 and 4 where we also underline the most relevant differences among them. Finally a glossary of all terms that are used in the MAS meta-model is reported in section 5.

2 PASSI and Agile PASSI Design Process

PASSI (Process for Agent Societies Specification and Implementation)[3, 6] is a step-by-step requirement-to-code design process conceived for developing multi-agent systems. It is characterized by two distinctive features: (i) it is requirement driven and (ii) it integrates contributes from both the object-oriented and agent-oriented paradigms.

PASSI is composed of five phases (or models) addressing different design levels of abstraction.
The **System Requirements Model** represents an anthropomorphic model of the system requirements in terms of agency and purpose. It consists of a functional description of the system to achieve: the designer identifies system requirements using use case diagrams, and then he separates the functionalities in packages. Then, the designer explores responsibilities through role-specific scenarios. In this phase the involved activities are not too different from other system requirement phases coming from traditional object-oriented processes. Hardly any term of the agent paradigm is used in this phase: system agents are yet identified but in this stage they are only an abstract entity, used to separate

responsibilities and, except the autonomy, no other classical agent features are referred, used or defined.

The next phase, the **Agent Society Model**, fully exploits the agent paradigm: now an agent is seen as an autonomous entity capable of pursuing an objective through its autonomous decisions, actions and social relationships. The activities of this phase aim to depict agent internal plans and social abilities in order to model interactions and dependencies among entities of the society. In the Ontology Description the designer depicts the domain ontology that is used to capture the domain complexity and to structure agents' knowledge and communications content. Then, in the Role Description the designer defines the life-cycle of each agent by looking at the roles it can play, at the collaboration that it needs, and the communications in which it participates. Finally, if necessary specific agent interaction protocols are designed to meet specific needs risen from previous activities.

The **Agent Implementation Model** defines the implementing details of the solution in terms of classes and methods. In this phase the designer uses conventional class diagrams, to describe the static structure of the involved agents, and activity diagrams or state charts, to describe the behaviour of individual agents. From these diagrams the programmer can directly produce the source code for a specific agent platform.

The **Code Model** is a representation of the solution at the code level while the **Deployment Model** describes the distribution of the parts of the system across hardware processing units, and their migration between processing units.

Agile PASSI [8, 7] derives from PASSI through the reuse of some of its parts and it has been assembled complying a method engineering approach [10–12]. It is a light process created, according to the agile manifesto [9], with the aim to front, in a short time, small-medium size systems. An agile process is easy to understand and to use because it is principally code oriented; for these reasons Agile PASSI comes to be well suited for those applications where coding is more important code than documentation. It is also useful when the programmer wants to reuse portions of design models and source code from other projects.

In order to be compliant with agile modeling principles [9, 13], Agile PASSI is an iterative process; it is composes of five steps (a low number) and it strongly involves users and customers during the development phases, especially during the planning one.

The fragments (portions of the design process) we have extracted from PASSI are: (i) *Domain Requirements description* (the description of system functionalities through use case diagrams), (ii) *Agent Identification* (the identification of logically related sets of functionalities that are put under an agent's responsibility), (iii) *Domain Ontology description* (the description of the agent knowledge in term of concepts, predicates and actions), (iv) *Code Reuse* (a technique for pattern reuse) and (v) *Testing* (single agent and society test).

This selection was done according to the PASSI philosophy; we have maintained use cases as the base for agents identification and we have not changed the fundamental role of the ontology in the process. From the other side we

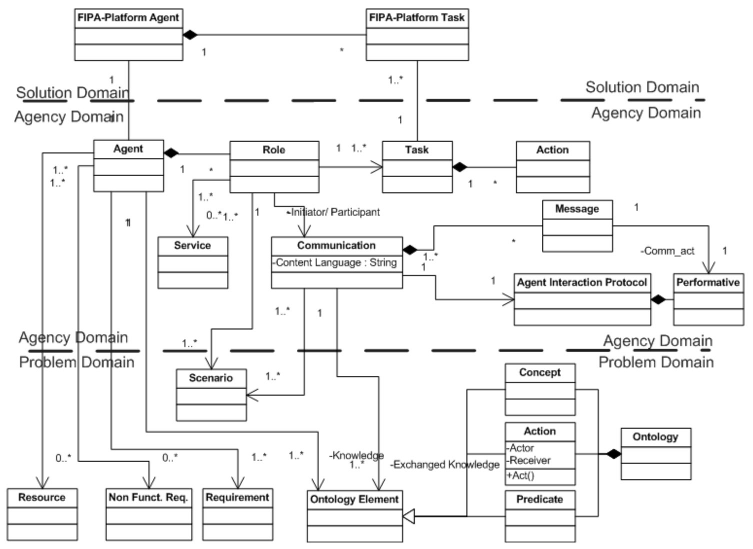

Fig. 1. The PASSI MAS meta-model

have respected the requirements for an agile process: low importance of high level documentation and rapid code production. The result of the composition of these fragments is a new process including five steps.

– **Requirements**, a model of the system requirements that is composed of two activities (Planning and Sub-Domain Requirements Description).
– **Agent Society**, a view of the agents involved in the solution, their interactions and their knowledge about the world. It is composed of two activities (Domain Ontology Description and Agent Identification).
– **Test Plan**, the phase of tests planning according to the eXtreme Programming rules [14].
– **Code**, a solution domain model at code level.
– **Testing**, the performing of the previous planned tests.

3 PASSI MAS Meta-Model

The description of the PASSI meta-model addresses three logical areas: (i) the problem domain, (ii) the solution domain and (iii) the agency domain; they are introduced in an order that reflects our choice of an agent approach for solution refinement and modeling.
In the problem domain we include components coming from the world where the software is going to operate: these are directly related to the requirements analysis phase of the PASSI process. Then we introduce the agency domain components; they are used to define an agent solution for the problem. Following this approach, we implicitly say that we see the agent paradigm as a problem decomposition and analysis instrument rather than a technological infrastructure for

systems implementation. This is motivated by the fact that most diffused agent-platforms [15][16] are just object-oriented development frameworks allowing the implementation of some agent-related concepts with a sufficient approximation. Obviously a pure agent-oriented solution would include an agent-oriented implementation language but despite of some existing experiences (like Jack [17], that is however still Java-based) we think technology is not mature for that. Just to provide an example of the features to be expected from a complete agent-oriented development language, we can consider inheritance that in a pure agent perspective should be interpreted in a way that is substantially different from the object-oriented one; it should focus on agent specific characteristics (like roles, communications, knowledge) rather than on methods and attributes.
Finally, in the PASSI MMM solution domain, agency-level components are mapped to the adopted FIPA-compliant implementation platform elements (we suppose the platform supports at least the concepts of agent and task); this represents the code-level part of the solution and it is the last refinement step.

Going into the details of the model, we can see that (Figure 1), the Problem Domain deals with the user's problem in terms of scenarios, requirements, ontology and resources. Scenarios describe a sequence of interactions among actors and the system (see section 5 for a complete definition of all MMM terms); they are used to identify the requirements that the system should fulfill. Requirements are represented with conventional UML use case diagrams. There is a strong point behind these choices: a lot of designers already skilled with such an approach are already present in different companies and can be more easily converted to the use of an agent-oriented methodology if they are already confident with some of the key concepts (and particularly the initial ones) used within it. The ontological description of the domain is composed of concepts (categories of the domain), actions (performed in the domain and effecting the status of concepts) and predicates (asserting something about a portion of the domain, i.e. the status of concepts). This represents the domain in a way that is near to the classic structural representations produced in the object-oriented analysis phase but it proves substantially richer than those. Resources are the last element of the problem domain. They can be accessed/shared/manipulated by agents. A resource could be a repository of data (like a relational database), an image/video or also a good to be sold/bought.

The Agency Domain contains the components of the agent-based solution. None of these is directly implemented; they are converted to the correspondent object-oriented entity that constitutes the real code-level implementation. In PASSI an agent is responsible for realizing some functionalities descending from one or more functional requirements and respecting some non functional requirement constraint (like for instance performance prescriptions). It lives in an environment from which it receives perceptions (the related knowledge is structured according to the designed domain ontology). Sometimes an agent has also access to available resources and it is capable of actions in order to pursue its own objectives or to offer services to the community.

Each agent during its life plays some roles. A role is a peculiarity of the social behavior of an agent. When playing a role, an agent may provide a functionality/service to other agents. The role concept is tightly related to the agent communication capability (interactions among agents are uniquely based on communications composed of messages seen as speech acts).
In PASSI, a task specifies the computation that generates the effects of a specific agent behavioral feature. It is used with the significance of atomic part for composing the overall agent's behaviour. This means that an agent's behaviour can be composed by assembling its tasks but the list of actions that are executed within each task cannot be influenced by the behaviour planning. Tasks are structural internal components of an agent and they contribute to define the agent's abilities; these cannot be directly accessed by other agents (autonomy) unless the agent offers them as a set of services. A communication is an interaction among two agents and it is composed of one or more messages. The information exchanged during a communication is composed of concepts, predicates or actions defined in the ontology. The flow of messages and the semantic of each message are ruled by an agent interaction protocol (AIP).
The last Agency Domain element (Service) describes a set of coherent functionalities exported by the agent for the community.

The Solution Domain describes the structure of the code solution in the chosen FIPA-compliant implementation platform (like FIPA-OS or JADE) and it is essentially composed of two elements: (i) the FIPA-Platform Agent that represents the base class catching the implementation of the Agent entity represented in the Agency domain; (ii) the FIPA-Platform Task that is the implementation of the agent's Task.

4 Agile PASSI MAS Meta-Model

Like the previous one, the Agile PASSI MMM (MAS meta-model) is partitioned in three logical areas: (i) problem domain, (ii) agency domain and (iii) solution domain. Agile PASSI was assembled starting from fragments extracted from PASSI, so the selection of MAS components was based on these fragments following a particular design process [7] also considering the particular applications the agile process was adopted to solve.

The Agile PASSI MAS meta-model is reported in Figure 2. An agile process principally addresses code production, so in this case MAS meta-model components are mainly centered on the agent and its related implementation parts. Using Agile PASSI a multi-agent system is conceived following four principal phases: planning, design, coding and testing; during the first two phases the Problem Domain MAS meta-model elements are defined, they are: functional and non functional requirements (they are used to describe the user point of view on the problem solution), and domain ontology (composed of concepts, predicates and actions). In this case requirements are not necessarily represented through use cases diagrams because the planning phase can also be carried out

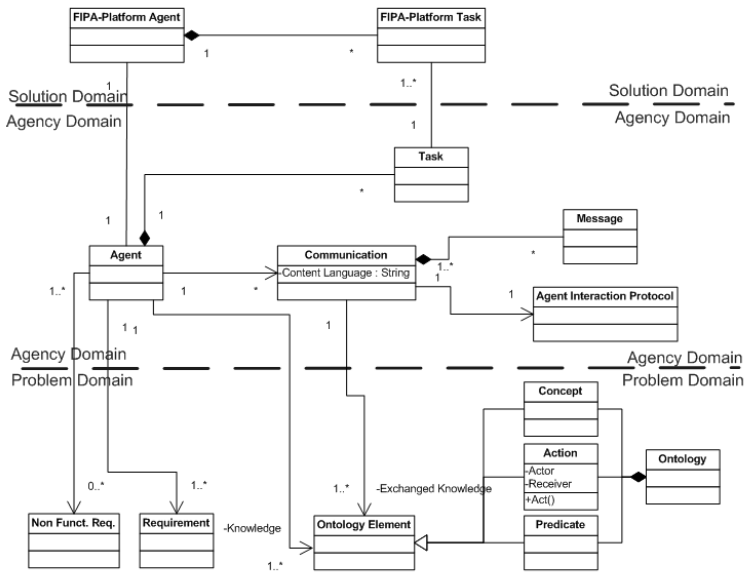

Fig. 2. Agile PASSI Meta-Model

informally (for instance orally), while ontology is necessary to model the agent's knowledge on the world and should be described as formally as in conventional PASSI.

As regards the Agency Domain, obviously the agent is its central component and it is conceived in the same way as it is in conventional PASSI; it is composed of tasks representing significant (but not divisible) parts of its behavior and its capability of pursuing an objective realizing system functionalities, besides it uses communications to interact (communicate or request collaboration) with other agents, each communication being composed of messages ruled by an interaction protocol (like it is in PASSI).

The Solution domain is exactly the same of conventional PASSI since it is composed of the *Agent* and *Task* implementation elements.

5 Glossary

In the following table a list of the elements used to compose the previous described MAS meta-models is presented with a description of their specific meaning.

Requirement	A requirement represents a feature that the system to be must exhibit, it can be a functional requirement such as service or a non-functional requirement such as a constraint on the system (or a specific part of it) performance.
Scenario	A concrete, informal description of a single feature of the system.
Service	A service is a single, coherent block of activity in which an agent will engage. A set of services can be associated with each agent role.
Resource	A concrete, tangible entity that can be acquired (also using sensors), shared, or produced by agents.
Agent	We consider two different aspects of the agent: during the initial steps of the design, it is seen (this is the Agency Domain Agent) as an autonomous entity capable of pursuing an objective through its autonomous decisions, actions and social relationships. This helps in preparing a solution that is later implemented referring to the agent as a significant software unit (this is the Solution Domain FIPA-Platform Agent). More in details, an Agent is an entity which: - is capable of action in an environment; - can communicate directly with other agents typically using an Agent Communication Language; - is driven by a set of functionalities it has to accomplish; - possesses resources of its own; - is capable of perceiving its environment; - has only a partial representation of this environment in form of an instantiation of the domain ontology (knowledge); - can offer services; - can play several different (and sometimes concurrent or mutually exclusive) roles.
Role	A portion of the social behaviour of an agent that is characterized by a goal (accomplishing some specific functionality) and/or provides a service.
Task	A task specifies the computation that generates the effects of the behavioural feature. Its granularity addresses the significance of a non decomposable group of atomic actions that cannot be directly addressed without referring to their belonging task.
Communication	An interaction among two agents, referring an Agent Interaction Protocol and a piece of the domain ontology (knowledge exchanged during the interaction). Usually it is composed of several messages, each one associated with one Performative.
Message	An individual unit of communication between two or more agents. A message corresponds to a communicative act, in the sense that a message encodes the communicative act for reliable transmission between agents. Each message It is related to a Performative.

Agent Interaction Protocol	A common pattern of conversations used to perform some generally useful task. The protocol is often used to facilitate a simplification of the computational machinery needed to support a given dialogue task between two agents.
Performative	It represents a verb that describes the action associated to a content sent to recipients. It carries the meaning of the message and what is the intention of the sender.
Ontology, concept, action, predicate	An ontology is an explicit specification of the structure of a certain domain Ontologies therefore provide a vocabulary for representing and communicating knowledge about some topic and a set of relationships and properties that hold for the entities denoted by that vocabulary.
FIPA-Platform Agent	The software implementation of the Agent in the selected (FIPA) platform
FIPA-Platform Task	The software implementation of the Task in the selected (FIPA) platform

References

1. Castro, J., Kolp, M., Mylopoulos, J.: Towards requirements-driven information systems engineering: The tropos project. In: To appear in Information Systems, Elsevier, Amsterdam, The Netherlands (2002)
2. Wooldridge, M., Jennings, N.R., Kinny, D.: The gaia methodology for agent-oriented analysis and design. Journal of Autonomous Agents and Multi-Agent Systems **3** (2000) 285–315
3. Cossentino, M., Potts, C.: A case tool supported methodology for the design of multi-agent systems, Las Vegas (NV), USA, The 2002 International Conference on Software Engineering Research and Practice, SERP'02 (2002)
4. DeLoach, S.A., Wood, M.F., Sparkman, C.H.: Multiagent systems engineering. International Journal on Software Engineering and Knowledge Engineering (**11**) 231–258
5. Capera, D., Georg, J.P., Gleizes, M.P., Glize, P.: The amas theory for complex problem solving based on self-organizing cooperative agents. In: Proc. of the 1st International Workshop on Theory And Practice of Open Computational Systems (TAPOCS03@WETICE 2003), Linz (Austria) (2003)
6. Cossentino, M.: From requirements to code with the passi methodology. In Henderson-Sellers, B., Giorgini, P., eds.: Agent-Oriented Methodologies, Idea Group Inc. (2005 (in printing))
7. Cossentino, M., Seidita, V.: Composition of a new process to meet agile needs using method engineering. In Ed., E., ed.: LNCS Series. (2004) 36–51
8. Chella, A., Cossentino, M., Sabatucci, L., Seidita, V.: From passi to agile passi : tailoring a design process to meet new needs. In: 2004 IEEE/WIC/ACM International Joint Conference on Intelligent Agent Technology (IAT-04), Beijing, China (2004)
9. Beck, K., al.M. Beedle, van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R., Mellor, S., Schwaber, K., Sutherland, J., Thomas, D.: (Agile manifesto) http://www.agilemanifesto.org.

10. Brinkkemper, S.: Method engineering: engineering the information systems development methods and tools. Information and Software Technology **37** (1995)
11. Kumar, K., Welke, R.: Methodology engineering: a proposal for situation-specific methodology construction. Challenges and Strategies for Research in Systems Development (1992) 257–269
12. Saeki, M.: Software specification & design methods and method engineering. International Journal of Software Engineering and Knowledge Engineering (1994)
13. Alliance, A.: (http://www.agilealliance.org)
14. Wells, D.: (Extreme programming - a gentle introduction) http://www. extremeprogramming.org.
15. Bellifemine, F., Poggi, A., Rimassa, G.: Jade - a fipa2000 compliant agent development environment. In: Agents Fifth International Conference on Autonomous Agents (Agents 2001), Montreal, Canada (2001)
16. Poslad, S., Buckle, P., Hadingham, R.: The fipa-os agent platform: Open source for open standards. In: 5th International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agents, Manchester, UK (2000)
17. Howden, N., Rnnquist, R., Hodgson, A., Lucas, A.: Jack summary of an agent infrastructure. In: Proc. of the 5th International Conference on Autonomous Agents. (2001)