Agile PASSI: An Agile Process for Designing Agents

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

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

Abstract. We have been developing robotic multi-agent systems for several years according to a well defined methodology (PASSI) obtaining good results, but day by day needs of a more versatile approach for designing software in a research context suggested us to find out a new methodology.

A solution to our problems is represented by the Agile version of the PASSI methodology we present in this paper. We built this agile methodology by exploiting all the experiences done with conventional PASSI; it is supported by specific tools allowing patterns reuse and automatic production of some design documentation.

1 Introduction

Robotic applications require a great attention toward domain specific problems like knowledge representation, environment exploration (with cameras, laser beams or other devices), actions planning, and coordination with other robots. During the development of such systems, researchers often prefer a prototypal approach to reach the solution in a short time. The greatest effort is spent in implementing complex algorithms so that not enough resources remain available for producing documentation. After all, it is reasonable that documenting intermediate prototypes cannot cost a lot of time while these are continuously changing. However, documentation is fundamental when components become more stable, otherwise their further maintenance will be very difficult.

Starting from the study of some recent experiences in software engineering (agile processes [1,2] and extreme programming [33]) we could remark that the motivations of this situation can be found in limits imposed by traditional software engineering design process: classical software engineering approaches are usually time consuming and the amount of produced documentation, although useful, is probably too large and detailed for the needs of several developers; moreover it is never easy to maintain different documents aligned with the changes.

In the past, we developed several robotic systems by using PASSI [13,11,10]; results were interesting [12] and the quality of design-related software attributes was remarkably high but the paradigm was not so fast and flexible as some developers would like to. One of the main critics we registered was related to some kind of anxiety that was induced in stakeholders involved in the process while producing the diagrams of initial iterations; they rather would like to have a more

direct way to experiment some code-level aspects of their application (for example they usually aimed at soon implementing algorithms characterizing the new solution for their specific problem).

In order to encompass these limits, we decided to produce an agile methodology; we also wanted to maintain our background of experiences accumulated in these years of designing systems with the PASSI methodology. The result of our work is an agile version of PASSI. To conceive it, we took advantage of studies about agent-oriented meta-methodologies [20,21] that starting from the method engineering approach (born in the object-oriented context [7,26,31]), allow the composition of a new methodology by reusing fragments of existing ones [17].

The paper is organized as follows: in the next section we will discuss our work by presenting its theoretical background and the specific needs we identified for robotic systems development process. In section 3 the previous presented needs will be used to deduce the strategic choices that define the skeleton of Agile PASSI that is then presented, more in details, in section 4. Experiences obtained by applying the new methodology are described in section 5 and some conclusions are drawn in section 6.

2 The new methodology requirements and composition process

When studied the solutions we are now presenting, we considered a specific problem: the need for a rapid development of a robotic application with a limited concern for the quality of the design and its documentation. This brought us to identify the need for an agile methodology supported by some design tool. Taking profit of our previous experience with the PASSI methodology [16,15], patterns reuse [12,9], and related design tools [13,30], we conceived an agile version of PASSI by reusing some parts of it and building up some new required portions of process, following the method engineering paradigm [5]. Our specific approach will be discussed in subsection 2.2 where we will point out several differences between the classical, object-oriented method engineering paradigm and its application to MASs (Multi-Agent Systems).

2.1 Requirements of our robotic design methodology

Our systems are deployed on mobile robots moving at a relatively low speed and usually performing missions related to the use of cognitive capabilities (for instance we designed a museum guide system [11], surveillance and environment discovery applications).

Our primary requirement is to allow a quick development, and in so doing we aimed at not distracting them with a long design process. This does not mean that we could accept a straight coding approach since: (i) our applications rapidly grow up in dimension and (ii) we have a specific concern about documenting the know-how reached in our laboratory in order to deliver it to new graduate and Ph.D. students that will collaborate in our future researches.

Another concern is related to the possibility of quickly reusing contributions coming from other projects; in this way the effort concerning the development of new applications may be restricted to the solution of their novel aspects only.

A further requirement of our methodology regards modelling real-time aspects of the developed systems; our time constraints are not very tight but nonetheless the possibility of explicitly designing concurrent actions is highly desirable in order to optimize the performance of a system. This is particularly important when using low efficiency agent platforms (Java-based) that could otherwise suffer of an unacceptable decay in performance.

We think that all of these issues could be satisfied by: (i) using an agile process (see sub-section 2.1.1) that supports a light (manual) design phase encouraging the reuse of existing contributions in form of patterns and (automatically) produces a consistent documentation at different level of abstractions. This methodology has to be supported by a design tool in order to

limit all manual operations that contribute in slowing down the process and could introduce mistakes in the final result.

In the following we will introduce the features of agile processes and our specific methodology construction process based on the method engineering paradigm.

2.1.1 Agile Design Processes

Lightweight methodologies, today called agile methodologies [2,1], arose in the last years as a reaction to the classic software development methodologies. The most important difference between the two kinds of methodologies is the quantity of documentation produced; classic methodologies rationale induces a large quantity of high level documentation while an agile methodology is principally based on code production, being the same source code the key element of documentation.

Agile methodologies principally follow four focal values [1, 22]:

- Individuals and interactions over processes and tools;
- Working software over comprehensive documentation;
- Customer collaboration over contract negotiation;
- Responding to change over following a plan.

The main aspects of agile methodologies are (i) simplicity and speed, (ii) incrementality and iteration, (iii) quick releases of prototypes, (iv) promotion for cooperative work (developers and customers collaborate with close communications), (v) easiness to learn, to modify and to face changes.

The iterative development allows continuous releasing of working components (prototypes) that will converge towards the final configuration; in each iteration this approach provides a base on which designers can plan next increments. Rapid prototyping aims at realizing software through successive product releases involving both client and users in the definition of the requirements thus avoiding misunderstanding and frequent changes. The prototyping approach proved to be a powerful way to take under control continuous changes, modelling through little increments, producing working portion of code as soon as possible, and then iterating to include other features.

One of the most used agile methodologies is eXtreme Programming (XP) [33], which is particularly suitable for small teams developing systems where requirements are vague and continuously changing. Knublauch, in his work [24,25], adopted this approach; he presented a convincing case study where he showed XP programming to be a good approach for multi-agent systems design and development.

Differently from Knublauch, we decided to build a new agile methodology that was specific conceived for our purposes; our main objective was not only to identify/create an agile methodology but we had several specific requirements to respect. XP was not a good solution for us because it violates at least one of our requirements: the reuse of experiences and skills we matured with conventional PASSI.

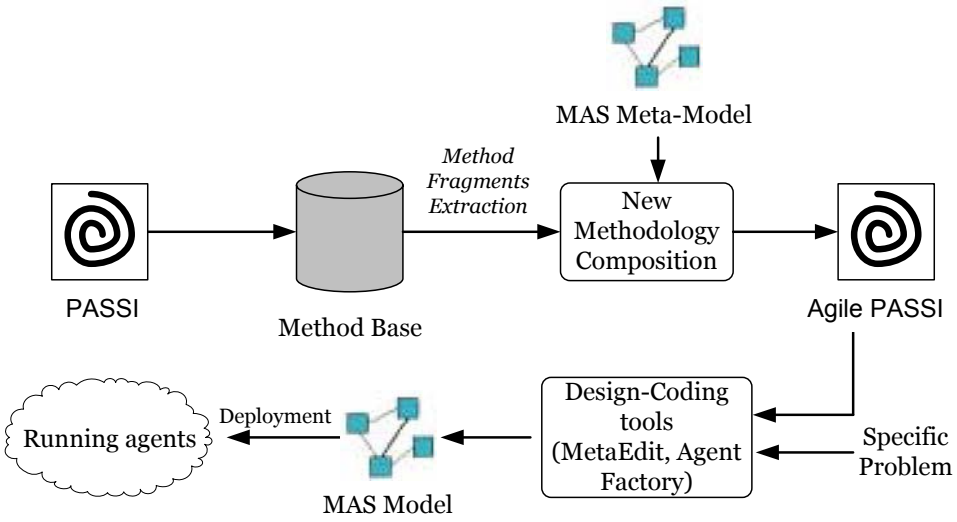

Figure 1. The Agile PASSI construction process

2.2 The methodology construction process

In order to build our new design process we referred to the experiences done with existing methodologies (and specifically with PASSI) and we adopted the method engineering paradigm [7,26,31]. According to this approach, the development methodology is built by assembling pieces of existing processes (called method fragments) [5,6,29] from a repository of methods. In this way we could obtain the best process for our specific needs.

In this work we consider a method fragment as composed by (see also the FIPA Methodology Technical Committee method fragment definition [19,27]):

- A portion of process (describing what is to be done in the specific fragment)
- One or more deliverables (artefacts like UML/AUML diagrams, text documents and so on).
- Some preconditions (like the required input data or the guard condition that is to be verified before starting the work specified in the fragment).
- A list of concepts (related to the MAS meta-model) to be defined/designed/refined by executing the specific method fragment.
- Guideline(s) that illustrates how to apply the fragment and best practices related to that.
- A glossary of terms used in the fragment.
- Other information (composition guidelines, platform to be used, application area and dependency relationships useful to assemble fragments).

We chose this approach because, in the last years, it proved successful in developing many object-oriented applications, for example information systems [32], and it is now collecting a growing interest from the agent community [21,23]. However, some relevant differences exist between the approach we used in building Agile PASSI and the similar ones adopted in the object-oriented (OO) context; the most relevant difference is that in the OO world a common denominator is the universally accepted concept of object and related model of the object oriented system while in the agent context there is not a univocal definition of agent nor any diffused model of the multi-agent system exists. Sometimes we can see that these concepts, for example the agent behaviour, are used, by different authors and in different methodologies, with slightly distinct meanings or granularity [3]. For this reason our work starts with the definition of a MAS meta-model; with the term MAS meta-model we mean a structural representation of the elements (agent, role, behaviour, ontology,…) composing the actual system together with their structural relationships.

Agile PASSI has been constructed according to the process described in Figure 1: before building the new methodology, we defined the MAS meta-model and then extracted all the method fragments from the PASSI methodology [14]. In defining and assembling our fragments we used a CAME tool (Computer Aided Method Engineering tool, MetaEdit+ by Metacase in our experiment) that offers a specific support for the composition of a methodology from existing fragments or by creating new ones.

The composition and selection of method fragments were done by pursuing two main goals: (i) including all the fragments needed to design the elements listed in the MAS meta-model and (ii) producing an agile process that could fulfil the already discussed requirements for a robotic oriented design process; the result will be described in section 4.

The MAS meta-model adopted in Agile PASSI is represented in Figure 2. Its elements are clustered in three groups: the Problem Domain elements dealing with the user's problem in terms of requirements and ontology; the Agency Domain elements addressing the agent-based solution to the problem described in the previous domain; the Implementation Domain elements describing the structure of the code solution in the chosen FIPA-compliant implementation platform (we always refer to FIPA-compliant systems in our work).

More in details, in Figure 2 we can see that the concept of agent represents the entity performing the system functionalities; agents' functionalities descend from requirements elicited during meetings with clients, users, developers and designers; they are represented using a conventional UML use case diagram. Agent knowledge is described in terms of instances of the domain ontology, which is a composition of concepts (entities and categories of the domain), predicates (assertions about elements of domain) and actions (that agents can perform in the domain affecting the status of concepts). In Agile PASSI we think an agent to be composed of tasks representing a portion of its behaviour and embodying its capabilities of pursuing a specific goal (related to the requirements to be fulfilled). An agent uses communications to realize its social relationships and to ask for collaborations from other agents. Each communication is composed of messages expressed in an encoding language and refers to an element of the ontology, besides the flow of messages is ruled by an interaction protocol (AIP).

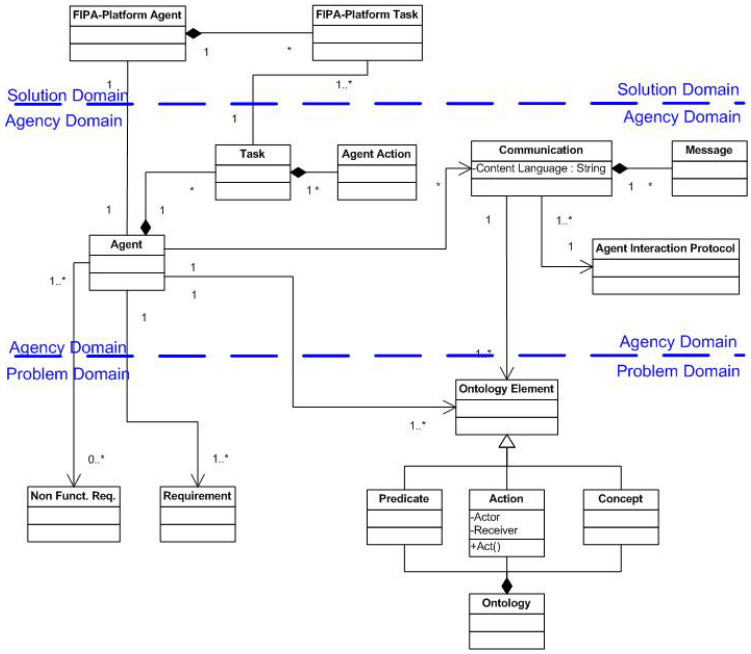

Figure 2. The multi-agent system meta-model adopted in Agile PASSI

3 The Agile PASSI Architecture

In this section we start from the analysis of PASSI and then by considering the requirements for the new methodology described in subsection 2.1, we select some of its fragments and use them to assemble the new agile methodology.

3.1 PASSI description

PASSI (Process for Agent Societies Specification and Implementation) [16] drives the designer from the requirements analysis to the implementation phase during the construction of a multi-agent system. The design work is carried out through the construction of five models obtained by performing twelve sequential and iterative activities. Briefly the phases and activities of PASSI are:

1. **System Requirements**. It is composed of four different activities and produces a description of the functionalities required from the system and an initial decomposition of them according to the agent paradigm. The four activities are: (i) the Domain Requirements Description, where the system is described in terms of the required functionalities; (ii) the Agent Identification where agents are introduced and the already identified requirements are assigned to them; (iii) the Role Identification where agents' interactions are described by using traditional scenarios; (iv) the Task Specification where the plan of each agent is draft.
2. **Agent Society**. It is composed of four activities producing an ontological view of the domain and the specification of agents' social interactions. In the Domain Ontology Description activity the the system domain is represented in terms of concepts, predicates, actions and relationships among them. In the Communication Ontology Description activity the focus is on the agents' communications that are explained in terms of referred ontological elements, content language and protocol. In the Role Description activity the distinct roles played by agents in the society and the tasks involved in playing each role are detailed.
3. **Agent Implementation**. It is a model of the solution architecture in terms of required classes with their attributes and methods. It is composed of two main streams of activities (structure definition and behaviour description) both performed at the single-agent and multi-agent levels of abstraction.
4. **Code**. It is a model of the solution at the code level. It is largely supported by patterns reuse and automatic code generation.
5. **Deployment**. It is a model of the distribution of the parts of the system across hardware processing units. The Deployment Configuration activity describes the allocation of agents in the units and any constraint on migration and mobility.

Testing has been divided into two different steps: the Agent and the Society test. In the first one the behaviour of each agent is verified with regards to the original requirements; during the Society Test, integration verification is carried out together with the validation of the overall results of the iteration.

This great number of steps may take a long time to obtain a first prototype. Also, the methodology is iterative both among the models and in the whole life cycle; this configures PASSI as a traditional methodology in which the coding phase is positioned somehow late in the process.

3.2 From PASSI to Agile PASSI: strategies and rationale

In subsection 2.1 we already presented the needs that arose from our experience in designing robotic systems, we now want to link them to the strategic choices that defined the skeleton of Agile PASSI. The identified requirements and consequent decisions are:

- The need for a short process, devoted to the frequent delivery of code; this logically brought us to conceiving an agile process.
- The need for the production of a documentation that could allow a good transfer of acquired knowledge in our laboratory; as a consequence we decided not to cut too much the different views proposed by PASSI but rather to obtain most of them by some kind of reverse engineering process.

The specific nature of the documents we decided to support arises from the dimension of the systems we usually produce; we think that three different representations are necessary: (i) a structural view of the system at the multi-agent (social) level of abstraction (it will be called MASD, Multi-Agent System Definition), (ii) a similar view representing the finer grained single-agent details level (whose name is SASD, Single-Agent System Definition), (iii) a representation of communications in order to depict social relationships and to let the designer follow the flow of information among agents. All of these diagrams should be zero-cost and therefore produced by an automatic tool. The need of representing time constraints and concurrent executions has been faced by a behavioural description of the system agents in form of activity diagrams (MABD, Multi-Agent Behaviour Description).

The opportunity of reusing the growing number of experiences, algorithms and parts of projects, strengthened the role that patterns reuse already played in conventional PASSI.

Lastly, we decided to take advantage of our experiences with PASSI by reusing a couple of its features that we consider very successful: (i) the identification of agents as a set of functionalities expressed in form of use cases (this activity is performed quite early in the process), and (ii) the central role of the domain ontological description in analyzing the agent solution.

All of these arguments brought us to identify the parts of PASSI (method fragments) that could be reused (or adapted) for the new methodology; after a detailed analysis we resolved that mainly five PASSI activities should be selected: Domain Requirements Description (DRD), Agent Identification (AId), Domain Ontology Description (DOD), Code Reuse (CR), Testing. We are now going to describe these fragments with more details.

As reported in sub-section 3.1, the PASSI methodology starts with a traditional system requirement analysis (called **Domain Requirements Description**, DRD). In this phase the designer explores the functionalities of the system drawing a hierarchical series of use case diagrams. During the following **Agent Identification** (AId) activity, use cases of the previous phase are grouped under the responsibility of different agents. The Agent Identification phase was maintained in Agile PASSI , because it allows an early designation of agents in the process; it allows thinking soon about agents rather than about some of their aspects (in some methodologies, for instance, roles are identified before the agents that will play them, this is a sort of bottom-up approach, we preferred a top-down one).

The **Domain Ontology Description** (DOD) is another fundamental phase in our methodology. While the AId is useful for exploring system functionalities, the definition of the system ontology allows a domain analysis at a conceptual level that easies the comprehension of the agent solution and produces a more accurate design. The elements of the system domain are identified and classified as concepts, predicates and actions. Concepts often become part of the agents' knowledge, while predicates and actions represent content data in communications.

These are the method fragments selected from conventional PASSI that can be reused without changes. According to XP main phases [33], we should now introduce the **Code Implementation** phase; this one, compared with the corresponding activity in the original PASSI methodology, arrives quite soon in the process. The available information for coding agents is, at this stage, less than it is in conventional PASSI and we recommend the adoption of as much patterns as possible for the rapid construction of a large part of the agent's structure. Patterns reuse is largely supported by Agent Factory [9]) whose main features are:

- Automatic completion from available design diagrams: the tool analyzes the Agent Identification and Domain Ontology diagrams and generates a first skeleton of the agent classes required for the implementation.
- Patterns Reuse: patterns may be introduced in the current project from a repository thus introducing new functionalities in a very low time and obtaining very affordable solutions.
- Automatic code generation: the results of the previous steps are weaved and the tool generates the code for the multi-agent system. This code consists in a skeleton of the agent and their task classes; the skeleton is completed by methods body coming from the reused patterns. Some experiments have shown a percentage of code reuse that is about 45-50% (one of these experiments is reported in [12]). Remaining parts of the code have to be manually written by the programmer.
- Reverse Engineering: the tool can extract several information from the actual agents' code; this information can be used to compose structural diagrams (like the previously discussed MASD and SASD diagrams and the COD) and behavioural diagrams (like the MABD). While the first kind of artefacts is a very common result of a reverse engineering process, the other one can be obtained because the code contains all that is needed to represent the flow of information and the flow of control of the system in a MABD activity diagram.

It is worth to note that while Agent Factory was used to support the cited activities, design of diagrams is supported by an add-in of Meta-Edit+ (a commercial CAME tool) that we produced in order to complement its design capabilities with some automatic composition, model checking and design process abidance features.

The **Testing** phase plays a fundamental role in all the agile processes because it represents the only way of checking the correctness of the system and its adherence to requirements. A test suite, based on the J-Unit framework, completes our development scenario [8]. Test plans are prepared before the coding phase accordingly with specifications and Agent Factory can be used to generate the necessary driver and stub agents.

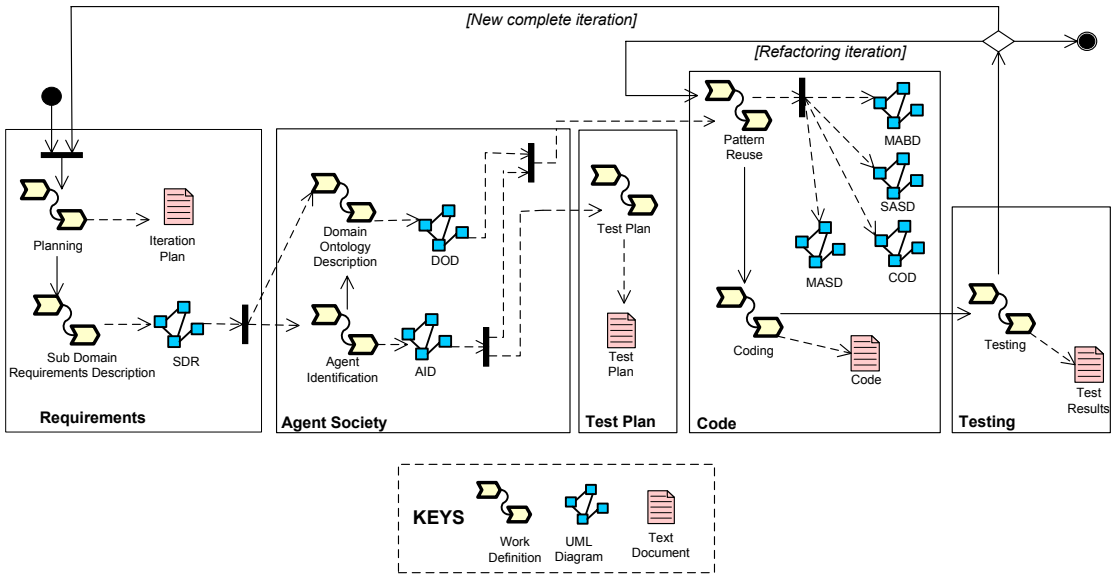

Figure 3. The Agile PASSI process

4 Agile PASSI description

Starting from the method fragments identified in the previous subsection and considering the requirements for the new methodology, we assembled the new Agile PASSI process described in Figure 1 using a SPEM (Software Process Engineering MetaModel, a specification by OMG) activity diagram [28]. There we can distinguish four phases:

- **Requirements:** a model of the system requirements that is composed of two steps (Planning and Sub-Domain Requirement 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 steps (Domain Ontology Description and Agent Identification).
- **Code**: a solution domain model at the code level.

  Testing: decomposed in Test Plan before coding and Testing soon after it.

According to the UML profile for SPEM specification, in Figure 3 we used three different icons to represent activities to be done and artefacts (UML models or text documents) to be produced in the process.

4.1 Requirements Model

It is composed of two activities: Planning and Sub-Domain Requirements Description. In the first one we plan one or more iterations privileging communications among development team components, through a risks and requirements analysis; it is performed, for instance, with the aid of so called user stories, similar to scenarios typical of XP programming. During this activity the development team decides which requirements of the system have to be faced and which in order this should be done.

In the Sub-Domain Requirements Description common UML use case diagrams are used to represent a functional description of the system. The term *sub-* refers to the opportunity of dividing

the entire problem in sub-problems. The result is a decomposition of the problem in several easier problems that will be faced in sequential iterations.

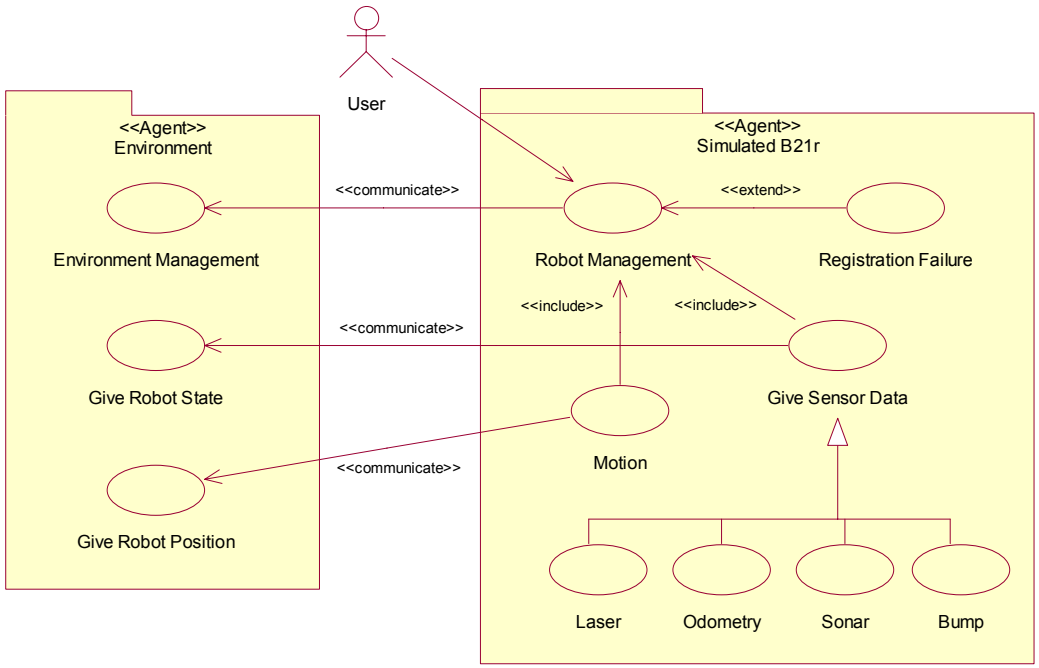

Figure 4. The Agent Identification Diagram designed in the first iteration for the experiment reported in section 5

4.2 Agent Society Model

Developing this model involves two activities: Agent Identification and Domain Ontology Description. The first activity gets as input the selected use case diagrams for the current iteration: here use cases have to be partitioned in separate groups of responsibility, (the agents of the system); in the diagram this means grouping one or more use cases into packages stereotyped with the *agent* label. An example is reported in Figure 4; it will be detailed in section 5.

The Domain Ontology Description activity aims to capture the domain in which the system will work identifying entities as UML classes. The ontology is described (see Figure 5) in terms of concepts (fill colour: yellow), predicates (fill colour: light blue) and actions (fill colour: white). Elements of the ontology can be related using three UML standard relationships: generalization, association and aggregation.

We think that these two activities should be performed in parallel, in fact, while deciding which functionalities will be assigned to different agents we could also identify and represent their knowledge.

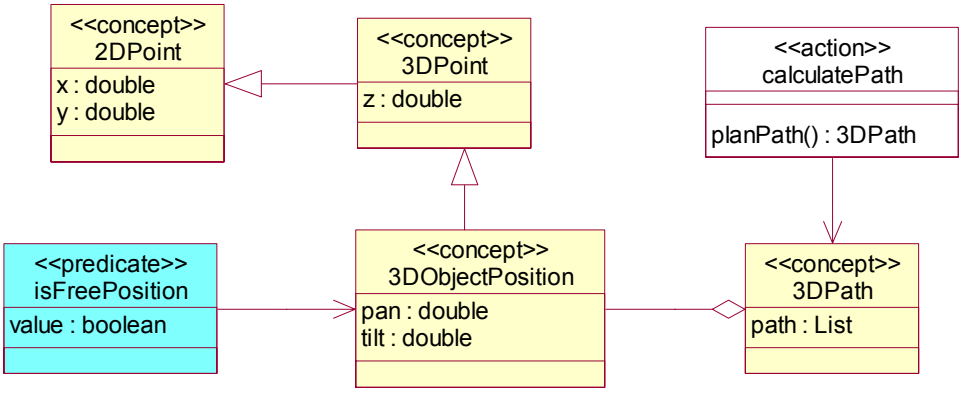

Figure 5 A Domain Ontology Diagram representing a portion of the ontology for the *Vision* agent represented in Figure 4

4.3 Code Model

This model includes two activities: Patterns Reuse and Coding. In the first one designers try to reuse design patterns of agents thus obtaining pieces of reusable code that is documented with both a structural and a behavioural view. This is done with the aid of Agent Factory [9], a tool we already adopted in the conventional PASSI. It allows the creation of a multi-agent system by reusing the functionalities of a pattern repository; it provides a support for the automatic compilation of a relevant amount of code (not only class skeletons but also inner parts of methods that are specified in the reused patterns) and it performs the reverse engineering of the manually modified code. This pattern-based approach improves project quality, increases the quantity of rapidly produced code and cuts down the overall time and costs of development [13].

In order to produce a minimal documentation of the design phase, we specifically produced an add-in for the MetaEdit+ tool. This add-in transforms information stored in the Agent Identification diagram and in the structural and behavioural models, generated by Agent Factory, into four kinds of documents:

- COD - a class diagram representing agents, their communications and related parameters (content language, agent interaction protocol and referred ontology);
- MASD - a class diagram where we represent the whole system at the social, multi-agent level of abstraction. It represents each agent with one class and agent's tasks as methods of the class;
- MABD - an activity diagram representing the flow of control and the communications among all the agents;
- SASD - one different class diagram for each agent in order to represent its internal structure and all of its tasks in the most detailed way.

In the Coding step programmers manually complete the previously produced code by putting in practice all the rules of extreme programming.

4.4 Test

The testing phase envelops the coding phase, it is divided in two parts (Test Plan and Testing) occurring before and after coding. According to eXtreme Programming (XP) rules test phase starts before coding and it should be continuous; designer and programmers plan one or more

tests that the agent must satisfy when coded. After the code is completed, during Testing phase, the actual test is performed accordingly to the previously defined test plans.

This represents a way to take under control programmers' work: if at least one test fails the component will be subject to a refinement and a refactoring until all tests are satisfied. When the test phase terminates successfully a working version of the agent is released. This may not be a final release but it is perfectly running, and it may be used as a prototype for a demonstration to the client.

5 Evaluation and Experimental Results

The evaluation of a methodology is a complex issue which can be faced from several points of view. In this work we examine the performance of Agile PASSI in two different ways: i) a qualitative exam is carried out by considering its proximity to some features that an agent-oriented methodology should have, ii) a quantitative evaluation has been done by repeating some projects (or better significant parts of them) that we already realized with conventional PASSI and by comparing the results in terms of development time and support received from adopted tools.

Dam and Winikoff in [18] present a list of categories deeply affecting the quality of a MAS design methodology. In this work they compare some methodologies by using a questionnaire that has been issued to designers experienced with those approaches and the authors of the methodologies. The basic criteria they consider (and we use for evaluating Agile PASSI) are:

- Concepts and ideas the methodology deals with.
- Models drawn and notations used to express them.
- The phases and steps that are part of the methodology (i.e. process aspect of the methodology).
- A range of practical issues that are concerns when adopting a methodology (pragmatics).

Considering the MAS concepts addressed in Agile PASSI, we can remark that it supports: (i) concurrency (it was one of the requirements identified in sub-section 2.1), (ii) multi-agent planning (represented in the MABD, Multi-Agent Behaviour Description, diagram), (iii) communications (detailed in all of their most important aspects) and (iv) environmental representation in terms of the knowledge that the agent achieves about it. Agents are also supposed to be autonomous and proactive (they will act according to their own plan to reach their goals without any supervision). On the contrary, no kind of description exists about mental attitudes (like believes, desires, and intentions). About the definition of specific (or proprietary) terms and their semantics we can say that the number of sources about conventional PASSI significantly contributes to define all of them and therefore this can be listed among the positive aspects too.

As regards modelling and notational aspects, Agile PASSI largely refers to UML extending it only to represent specific issues of agency that could not be coped with an object-oriented notation; again the existing documentation about PASSI (that is even defined according to OMG SPEM specifications [28]) definitely contributes to clarify everything. Obviously the use of our notation will be easy to understand only for designers already skilled with UML and object-oriented design.

Traceability is one of the major advantages of Agile PASSI: the add-in we developed for Meta-Edit and the patterns reuse/reverse engineering tool (Agent Factory), give a decisive contribution in this direction by verifying the consistency among design artefacts and corresponding code at different stages of the process.

Our process is iterative, composed by a low number of steps and strongly involves the end-user (or customer). This is a precise choice done to be compliant with the agile manifest principles [2]. After an initial phase of study and planning the process may be resumed in the following

phases: Test Planning - Code Reuse - Programming - Testing - Refactoring. As a consequence some of the phases of traditional methodologies are not considered or performed very quickly. Quality assurance is pursued by largely reusing patterns and automatically producing relevant portions of code. No support is provided for management issues or estimation about time, costs and so on.

Considering the methodology pragmatics we should note that Agile PASSI is conceived to be used in our laboratory by graduating students and researchers in accordance with the requirements presented in sub-section 2.1. We adopted the rationale of agile methodologies because it perfectly fitted in our needs (students and researchers skilled in code production with no enough time to produce high quality documentation). The reuse of some fragments helped in minimizing the cognitive effort needed for conventional PASSI experienced users to move from one methodology to the other.

Almost all the steps of the process are supported by automatisms and tools. Frequently these tools are used to automatically complete or partially compile the prescribed diagrams; this is obtained with a reverse engineering process that reuses the information already introduced by designers/programmers in the previous phases. Some limits still exist in the usability of all of these tools; they are totally integrated but they could create some problems if specific steps are not performed in the right way; moreover, we are still experiencing some troubles in redesigning some diagrams in successive iterations (problems are related to the position of diagram elements that are not properly located with respect to the others after a refresh operation).

Other pragmatic considerations can be done by comparing PASSI and Agile PASSI in terms of their structures (phases/activities) and in term of the number and kinds of the work products they specify. Results of this comparison are drawn up in Table 1. Here we show all the activities the designer carries out within each phase of the two methodologies; we also report the work products produced in each of these activities. Each PASSI phase/activity is put in relationship (in the same row) with the corresponding Agile PASSI element, this allows an easy identification of the activities and work products that were added, maintained and removed in creating Agile PASSI.

For instance we can see that the PASSI “System Requirement” phase corresponds to the Agile PASSI “Requirements” phase, here we added the “Planning” activity and we did not include the PASSI RID (Role Identification) and TSP (Task Specification) activities. The PASSI DRD (Domain Requirements Description) has been modified in SDRD (Sub-Domain Requirements Description) to comply with the agile philosophy of performing a requirements description in an iterative/incremental way. We can also note that the number of phases in Agile PASSI is only a little smaller than in conventional PASSI (six to four) while the number of activities is very different (about the half); this makes Agile PASSI a shorter process.

Table 1. Comparison among phases/activities and work products of PASSI and Agile PASSI

PASSI			Agile PASSI		
Phases	**Activities**	**Work Products**	**Work Products**	**Activities**	**Phases**
System Requirement	---	---	M,1	Planning	Requirements
	DRD	M,1	M,1	SDRD	
	AID	M,1	M,1	AID	
	RID	M,1..scenarios	---	---	
	TSP	M,1..agents	---	---	
Agent Society	DOD	M,1	M,1	DOD	Agent Society
	COD	P,1		---	
	RD	F,1	---	---	
	PD	M,0..n	---	---	
Agent Implementation	MASD	F,1	---	---	---
	MABD	P,1	---	---	
	SASD	P,1..agents	---	---	
	SABD	M,1..agents	---	---	
Code	CR	P,1	COD (F,1) MASD (F,1) MABD (F,1) SASD (F,1..agents)	Pattern Reuse	Code
	CP	F,1	Code Reuse (F,1) Code (M,1)	Coding	
Deployment	DC	M,1	---	---	
Testing	Agent Test	M,1..n	Test Plan (M,1..n)	Agent Test	Test
	Society Test	M,1	Test Plan (M,1)	Society Test	

For each work product in Table 1 we also specify if it is manually drawn by the designer (M), if it is partially compiled by the tools (P) or if its production is fully automated (F); we also reported the expected number of instances of the specific kind of diagram (for instance RID diagrams, that are sequence diagrams used to depict scenarios in conventional PASSI, are produced in a number that is equal to the considered scenarios).

From this information we can deduce that the Agile PASSI work products are about two thirds of the PASSI ones and that while only one sixth of artefacts in PASSI are fully automated this value raises to one half in Agile PASSI.

In order to obtain an estimation of the time effort required for developing a robotic application with Agile PASSI we performed some experiments. One of these consists in developing a system controlling the navigation of a Koala robot in an unknown environment. The obstacle avoidance behaviour is supported by the on board IR sensors and by an eye-bird camera looking at the environment. Cognitive aspects about the discovery of new obstacles are among the requirements of such an application. This is a typical case study in a robotics base class.

The multi agent system produced was composed by seven agents running in three computers connected by a LAN. The total dimension of the application was more than five thousands lines of code (this is not a big amount of code but agile approaches specifically address small to medium sized projects). Figure 5 we produced during this experiment shows a piece of the ontology (DOD diagram) concerning the navigation on a Cartesian space; we identified a path (*3DPath*) as an aggregation of *3DObjectPosition* (that is a *3DPoint* with pan/tilt information). Base on these concepts defined a predicate (*isFreePosition*, stating if a point is free for navigation) and an action (*calculatePath*).

It is worth to note that during the reuse phase, the use of automatism, patterns and the automatic code generation produced a total reuse of approximately 45-50% of the entire code thus obtaining a significant reduction of the work for the programmer.

Another case study is the development of a robotic 3D simulator: the system simulates a B21r robot, and an indoor operating environment. Figure 4 shows the Agent Identification Diagram for the *SimulatedB21r* and *Environment* agents. Some of the functionalities concern the robot sensors: *Laser*, *Odometry*, *Sonars* and so on. *Motion* is another interesting *SimulatedB21r*'s use case that simulates the movement of the robot interactiing with the *Environment* agent for receiving and updating its position.

The development of this system was assigned to two independent teams composed by two graduated students each. They worked in different times but each group was not aware of the other. The code length was about 60 thousands lines specifying 8 different agents. The time for completing this application (without considering the study and tuning of algorithms) took 23 days for the whole PASSI project while rebuilding the application with Agile PASSI needed 16 days; the final release (in Agile PASSI) has been obtained after three iterations each of which terminated with a running prototype (usually one or two agents).

The documentation produced with this design experience is good in quality, consistent and sufficiently describes the system; as a matter of fact, however, it is less than the documentation produced with conventional PASSI and this partly justifies the difference in time. A more detailed comparison of the time spent by the designers in the different phases prescribed by the methodologies is reported in Figure 6. Reported values are normalized (time actually spent is divided by the total time). In the diagram we can see that a few Agile PASSI phases took no time (they are not supported or fully automated) while they needed a relevant time in PASSI; some other phases are shorter in Agile PASSI than they are in PASSI (like requirements description); this is justified by the fact that they face smaller pieces of the problem each time.

Finally some other activities took more time in Agile PASSI than in PASSI; this is the case of the Agent Identification phase; here Agile PASSI designers spent a shorter time than the PASSI ones. We were expecting for an opposite result; probably this happens because designers in this phase (mostly in early iterations) have an incomplete view of the entire domain and this makes it more difficult to find the best clustering of use cases within agents (this is an activity that requires a good understanding of the whole system functionalities). The same occurs in the Domain Ontology Description (DOD) phase. This time we think that the best exploration of the problem that PASSI designers do before the DOD phase speed up their modeling of the ontology.

The final significant difference is in the Coding phase where time employed changes sensibly: from 25% (PASSI) to 50% (Agile PASSI): this is in the nature of an agile approach where focus is more on coding (and testing) than designing.

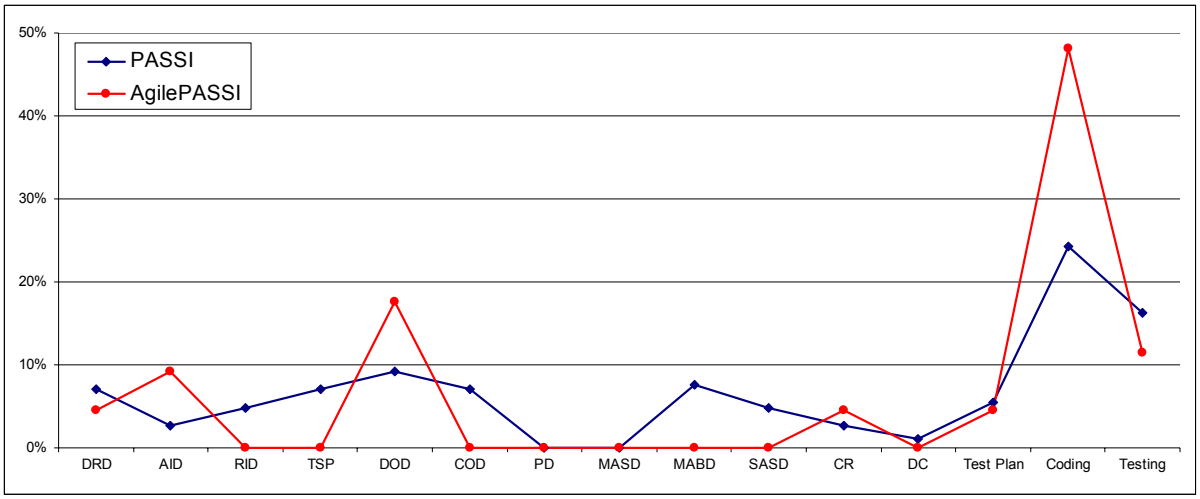

Figure 6. The amount of time (percentage) spent in the different phases of PASSI and Agile PASSI processes in our experiment

6 Conclusions and future works

In this paper we presented a new methodology, Agile PASSI that we conceived in order to have a design process that specifically addresses the needs of developing robotic systems. In the last years we adopted the PASSI design methodology and the results were good but, the length of the design process distracted developers from crucial algorithmic aspects that were faced too late; as a consequence we recently decided to look for a new, more versatile and quick process.

Agile PASSI is supported by an add-in that we produced for the design tool we adopted (MetaEdit+ by Metacase) and a patterns reuse/reverse engineering application that is a new evolution of the already presented Agent Factory. We developed a few systems with Agile PASSI and now we have a reasonable level of confidence with it; in this work, in order to evaluate its goodness, we reported a qualitative analysis based on the assessment of several attributes that should characterize an agent-oriented methodology and a quantitative one based on the comparison between PASSI and Agile PASSI in terms of activities, work products and developing time. We also described the results of two example applications that have been developed in both conventional PASSI and Agile PASSI.

In building our methodology we started from the analysis of our requirements, the study of agile processes and the method engineering approach; we think that this technique is general enough to be used for producing agile versions of other MAS methodologies when required.

In the future we plan of exploring the contemporary adoption of both PASSI and Agile PASSI in large projects. We think that thanks to the availability of two complementary design

processes sharing the same foundation, a team of developers can proceed along the classical PASSI design process while activating some minor iterations performed with Agile PASSI in order to deal with some critical or specific aspects of the system. Then results of these iterations can be easily integrated with the mainstream design thanks to the similarity in the most important artefacts produced by the two methodologies.

7 References

1. Agile Alliance [page on the Internet]. Available from: http://www.agilealliance.org.
2. Agile Manifesto. [page on the Internet]. Available from: http://agilemanifesto.org.
3. Bernon C, Cossentino M, Pavón J. An Overview of Current Trends in European AOSE Research. Informatica Journal (in printing).
4. Borenstein J, Koren Y. The vector field histogram - fast obstacle avoidance for mobile robots. IEEE Journal of Robotics and Automation; 1991; 7(3):278–88.
5. Brinkkemper S, Lyytinen K, Welke R. editors. Method engineering: principles of method construction and tool support. In: International Federational for Information Processing. Kluwer Academic Publishers; 1996; 65:336.
6. Brinkkemper S, Saeki M, Harmsen F. Meta-modelling based assembly techniques for situational method engineering. Information Systems; 1999; 24(3):209-28.
7. Brinkkemper S. Method engineering: engineering the information systems development methods and tools. Information and Software Technology; 1995; 37(11).
8. Caire G, Cossentino M, Negri A, Poggi A, Turci P. Multi-agent systems implementation and testing. Proceedings of Fourth International Symposium: From Agent Theory to Agent Implementation (AT2AI-4), Vienna, Austria (EU), April 14-16 2004.
9. Chella A, Cossentino M, Sabatucci L. Designing JADE systems with the support of CASE tools and patterns. Exp Journal; 2003; 3(3):86–95.
10. Chella A, Cossentino M, Fisco M, Liotta M, Rossi A, Sajeva G. Simulation based planning and mobile devices in cultural heritage robotics. Proceedings of the Robotics Workshop in the Ninth Conference of the Associazione Italiana per l'Intelligenza Artificiale (AI*IA), 2004 Sept, Perugia, Italy.
11. Chella A, Cossentino M, Pirrone R, Ruisi A. Modeling ontologies for robotic environments. In: Chang SK editor. Proceedings of the 14th international Conference on Software Engineering and Knowledge Engineering (SEKE '02), 2002 July 15-19; Ischia, Italy. ACM Press; New York, NY; 2002; 27(1):77-80.
12. Cossentino M, Sabatucci L, Chella A. A Possible Approach to the Development of Robotic Multi-Agent Systems. In: Liu J, Faltings B, et al. editors. Proceedings of the IEEE/WIC international Conference on intelligent Agent Technology (IAT); 2003, October 13-17. IEEE Computer Society; Washington DC; 2003; p. 539-44.
13. Cossentino M, Sabatucci L, Chella A. Patterns reuse in the PASSI methodology. In: Omicini A, Petta P, Pitt J, editors. Lecture Notes in Artificial Intelligence (LNAI); Springer; 2004; 3071:294-310.
14. Cossentino M, Sabatucci L, Seidita V. Method fragments from the PASSI process. ICAR-CNR technical report 2003; 21(03).
15. Cossentino M, Potts C. A case tool supported methodology for the design of multi-agent systems. Proceedings of the 2002 International Conference on Software Engineering Research and Practice (SERP), 2002 June 24-27, Las Vegas (NV), USA; p. 315-21.
16. Cossentino M, Sabatucci L. Agent System Implementation. In: Sacile R, Pauolucci M. editors. Agent-Based Manufacturing and Control Systems: New Agile Manufacturing Solutions for Achieving Peak Performance. CRC Press; 2004; April; p. 153-92.
17. Cossentino M, Seidita V. Composition of a New Process to Meet Agile Needs Using Method Engineering. Software Engineering for Large Multi-Agent Systems. Lecture Notes in Computer Science; Springer; 2005; 3:36-51.

18. Dam KH, Winikoff M. Comparing Agent-Oriented Methodologies. Proceedings of the Fifth International Bi-Conference Workshop on Agent-Oriented Information Systems (AAMAS). 2003; July 14; Melbourn, Australia. Lecture Notes in Computer Science; Springer; 2004; 3030:78-93.
19. FIPA Methodology Technical Committee [page on the Internet]. Method fragment definition. Nov 2003. Available from: http://www.fipa.org/activities/methodology.html.
20. Guessom Z, Cossentino M, Pavon J. Roadmap of Agent-Oriented Software Engineering: The European Agentlink Perspective. In: Bergenti F, Gleizes MP, Zambonelli F, editors. Methodologies and Software Engineering for Agent Systems. Kluwer; 2004; p. 431-50.
21. Henderson-Sellers B, editor. Creating a comprehensive agent-oriented methodology - using method engineering and the OPEN metamodel. In: Agent-Oriented Methodologies, Idea Group, 2005.
22. Highsmith J, Cockburn A. Agile Software Development: The Business of Innovation. Computer; IEEE JNL; 2001; 34(9):120-7.
23. Juan T, Sterling L, Winikof M. Assembling agent oriented software engineering methodologies from features. Proceedings of Third International Workshop on Agent-Oriented Software Engineering (AOSE), 2002; Bologna – Italy; p. 198-209.
24. Knublauch H, Koeth H, Rose T. Agile Development of a Clinical Multi-Agent System: An Extreme Programming Case Study. Proceedings of the Third International Conference on eXtreme Programming and Agile Processes in Software Engineering; 2002, May; Alghero, Sardinia, Italy.
25. Knublauch H. Extreme Programming of Multi-Agent Systems. Proceedings of the First International Joint Conference on Autonomous Agents & Multi-Agent Systems (AAMAS), 2002; Bologna, Italy. ACM Press; 2002; p. 704-711.
26. Kumar K, Welke R. Methodology Engineering: a proposal for situation-specific methodology construction. In: Cotterman W, Senn J, editors. Challenges and Strategies in Systems Development; 1992; p. 257-69.
27. O'Brien P, Nicol R. Fipa - Towards a standard for software agents. BT Technology Journal; 1998; 16(3):51–59.
28. OMG Object Management Group [page on the Internet]. Software Process Engineering Metamodel version 1.0; Nov 2002. Available from http://www.omg.org/technology /documents/formal/spem.htm.
29. Ralytè J, Rolland C. An approach for method reengineering. Lecture Notes in Computer Science; Springer; 2001; 2224:471-84,.
30. Sabatucci L, Cossentino M. A multi-platform architecture for agent patterns representation and reuse. Proceedings of Workshop from Objects to Agents (WOA) 2003; September 10-11; Villasimius (Cagliari) - Italy. Pitagora Editrice Bologna; 2003; p. 170-4.
31. Saeki M. Software specification & design methods and method engineering. International Journal of Software Engineering and Knowledge Engineering, World Scientific Pub. Co. ; 2002; 2:324-31.
32. Tolvanen JP. Incremental method engineering with modeling tools: Theoretical principles and empirical evidence (ph.d. thesis). Jyvskyl Studies in Computer Science 1998.
33. XP Extreme Programming. [page on the Internet]. Available from: http://www.extreme programming.org.