From PASSI to Agile PASSI: tailoring a design process to meet new needs

Antonio Chella
University of Palermo
Dipartimento di
Ingegneria Informatica (DINFO)
Viale delle Scienze, 90128 -Palermo- Italy
chella@unipa.it

Massimo Cossentino, Luca Sabatucci, Valeria Seidita
Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR)
Consiglio Nazionale delle Ricerche(CNR)
Viale delle Scienze, 90128 -Palermo- Italy
cossentino@pa.icar.cnr.it,
sabatucci@csai.unipa.it
seidita@csai.unipa.it

Abstract

From several years we are developing robotic multi-agent systems according to well defined design methodologies. These methodologies evolved over time because of the changes in the operating environments (robotic hardware and software platforms) and specific missions accomplished by our robots. In the last three years we used PASSI (Process for Agent Societies Specification and Implementation) obtaining good results but, the growing experience and day by day accelerating changes in requirements suggested us to find a new and more versatile approach. In this context we developed the Agile PASSI methodology discussed in this paper; it is an agile process built up capitalizing all the experiences done with PASSI and its supporting tools some of which have been adapted and reused in the new process.

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; the complexity of these issues often brings researchers in the field to devote a limited amount of time and effort to following a rigorous design process also if they are aware that it could produce an efficient documentation for further maintenance and better the quality of the result. Looking at most recent experiences in software engineering (agile processes [22][3] and extreme programming [16]) we could remark that some of the motivations of the above discussed situation can be found in the limits imposed by traditional software engineering design process. They are usually time consuming and the amount of produced documentation although useful is probably too large and detailed for the needs of several developers. In the past, we developed some robotic systems by using PASSI [23][11]; results were interesting[14] and the quality of design-related software attributes was remarkably high but the paradigm was not so fast and flexible as 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 the first iteration; they rather would like to have a more direct way to experiment some code-level aspects of the application (for example they usually aimed at soon implementing new algorithms characterizing their application).

In order to encompass these limits, we decided to produce and agile version of PASSI. In so doing we took advantage of studies about agent-oriented meta-methodologies[26][24][17] that starting from the method engineering approach born in the object-oriented context [7][21][27], allow the composition of a new methodology by reusing fragments of existing ones and, when necessary, introducing new, specifically created, parts in the process. In the next section we will discuss our work by presenting its theoretical background and the specific needs we identified for a robotic systems development process. In section 3 the previous presented needs will be used to make 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 Theoretical Background

In studying the solutions presented in this paper, we considered a specific problem, the rapid development of a robotic application accepting very low compromises on the quality of the design and its documentation. This brought us to identify the need for an agile methodology that could be

supported by some design tool. Taking profit of our previous experience with the PASSI methodology[13], patterns reuse[14], and related design tools[26][12][23], we conceived an agile version of PASSI by reusing some of its parts (called method fragments) and building up the new required portions of the process. This corresponds to applying the method engineering approach that will be discussed in subsection 2.3 to the composition of this process. Several differences exist in using the method engineering approach in its original field (object-oriented systems) and in MAS (multi-agent systems). All of these issues will be discussed in the following sub-sections.

2.1 Requirements of a robotic design methodology

Our systems are deployed on mobile robots moving at a relatively low speed (only a few meters per second) and usually performing missions related to the use of cognitive capabilities (for example we designed systems for museum guide, surveillance and environment discovery applications). Our primary requirement is related to not distracting developers from their main goal (tuning some kind of new algorithm) 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 students that will collaborate in our future researches. Another wish is related to the possibility of quickly reusing contributions coming from other projects in order to restrict the effort related to the development of a new application to the solution of its novelty aspects. Dealing specifically with robotics, this problem is less complex than it could seem since great parts of the system could be reused both from the algorithmic (general navigation solutions like path planning and obstacle avoidance) and structural (communications, resource sharing and data caching) points of view. As regards the response time of the developed systems, our real-time constraints are not very tight (as already said, ground robots move relatively slow) but nonetheless the possibility of explicitly designing concurrent actions and time relationships among them is highly desirable in order to optimize the performance of a system that because of the use of low efficiency agent platforms (Java-based) could otherwise bring to an unacceptable decay in performance. We think that all of these issues could be satisfied by: (i) using an agile process (see subsection 2.2) that supports a light (manual) design phase while encourages 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 operations done by hand that contribute in slowing down the process and could introduce mistakes in the final result.

2.2 Agile Processes

Classic software development methodologies are well disciplined and heavily oriented to make a process predictable and have a great stress on planning. As a reaction to this way of developing software,in the last years a new kind of methodologies, called lightweight in a first time but now known as agile, has been developed. An important difference between the two kinds of methodologies is the smaller quantity of documentation produced in the second case, in fact agile methodologies are code-oriented being source code the key element of documentation. The large quantity of high level documentation we create while performing a classic methodology induces some limitations in facing changes; a very powerful way to take under control continuous changes is modeling through little increments, producing working portion of code as soon as possible, and then iterating to include other features. The concept of iterative development has a fundamental consequence that is to continuously realize working subsystem that have not (yet) all the functionalities of the final system but when tested and integrated, they will provide the requested features. In each iteration, this approach provides a base on which we can plan the following increments. Finally we can say that agile methodologies are not complete methodologies but they are a supplement to the already existing ones, they begin where the other fault or better where the other needs changes in order to perceive their objective. Our attempt is, now, to reexamine PASSI, using principles and techniques of Agile Methodologies [22][3], in order to create a lightweight methodology, simple, easy to use and principally based on code production rather than on documentation (that is still requested, but when it can be automatically produced). In our work we followed the fundamental strategies of the Agile Manifesto: (i) Individuals and interactions over processes and tools, (ii) Working software over comprehensive documentation, (iii) Customer collaboration over contract negotiation, (iv) Responding to change over following a plan. We also considered the sequence of activities defined in one of the most used agile methodologies, Extreme Programming[16]: (i) Planning, (ii) Designing, (iii) Coding, and (iv) Testing. As it will be presented later, this sequence will constitute the center of the proposed methodology.

2.3 The methodology construction process

In order to build our new design process taking advantage of the experiences done with existing methodologies (and specifically with PASSI) we adopted the method engi-

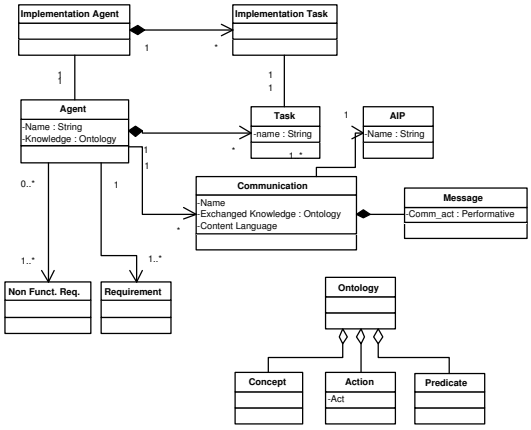

Figure 1. The multi-agent system meta-model adopted in Agile PASSI

neering paradigm [7][21][27]. According to this approach, the development methodology is built by assembling pieces of the process (method fragments) [25][8][9] from a repository of methods. In this way we could obtain the best process for our specific needs. We chose this approach because, in the last years, it proved successful in developing many object-oriented applications for example information systems [28] and is now collecting a growing interest from the agent community[18]. Some relevant differences exist between the approach we used in building Agile PASSI and similar approaches explored in the object-oriented context; the most relevant one is that in the OO context the construction of method fragments (pieces of methodology), the assembling of the methodology with them and the execution of the design rely on a common denominator, the universally accepted concept of object and related model of the object oriented system. In the agent context, there is not an universally accepted definition of agent nor it exists any very diffused model of the multi-agent system. Referring to a MAS meta-model we mean a structural representation of the elements (agent, role, behavior, ontology,...) that compose the actual system with their composing relationships. Sometimes we can see that these concepts, for example the behavior, are used, by different authors and in different methodologies, with slightly distinct meanings or granularity. We built Agile PASSI by referring the MAS meta-model represented in Figure 1. There the concept of agent represents the entity performing the system functionalities. Each functionality descends from one or more requirements elicited during meetings with clients, users, developers and designers and then represented in a conventional use case diagram. Agent knowledge is described in term of instances of the domain ontology, that 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, so affecting the status of concepts). In Agile PASSI we think to an agent as composed of tasks representing a portion of its behavior and embodying its capabilities of pursuing a specific goal. An agent uses communications to realize its social relationships and asking 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).

Agile PASSI has been constructed according to the process described in Figure 2: before building the new methodology, we selected the elements that compose the meta-model of the MAS we will build and extracted all the method fragments from the PASSI methodology [5][4]. In this work we consider a method fragment as composed by (see also the FIPA method fragment definition [2]):

1. A portion of process
2. One or more deliverables (artifacts like (A)UML/UML diagrams, text documents and so on). Some preconditions (like the required input data or guard condition)
3. A list of concepts (related to the MAS meta-model) to be defined/designed/refined by executing the specific method fragment.
4. Guideline(s) that illustrates how to apply the fragment and best practices related to that A glossary of terms used in the fragment
5. Other information (composition guidelines, platform to be used, application area and dependency relationships useful to assemble fragments) complete this definition.

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 has been 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 fulfill the already discussed requirements for a robotic oriented design process; the result will be described in section 4.

3 Agile PASSI Architecture

In this section we will start from the analysis of PASSI (that is our reference methodology) and then by considering the requirements for the new methodology described in

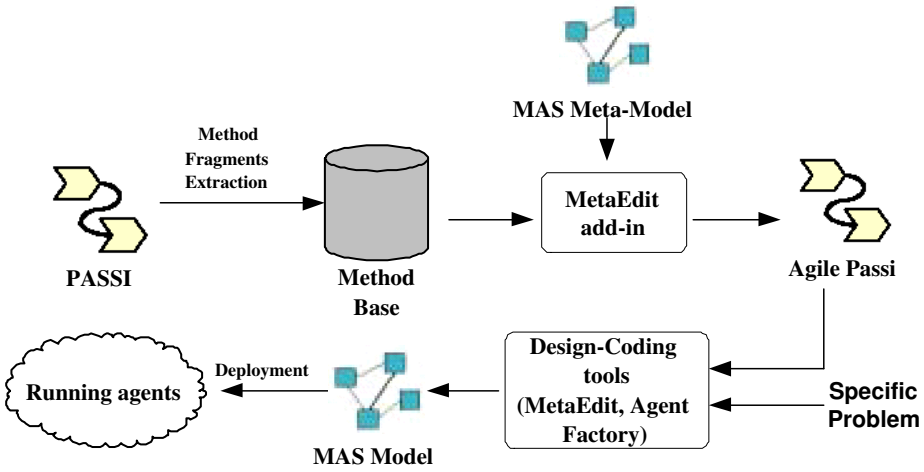

Figure 2. The Agile PASSI construction process

subsection 2.1 we will select some portions of PASSI (i.e. method fragments) and then use them to assemble the new agile methodology.

3.1 PASSI description

PASSI drives the designer from the analysis requirement to the implementation phase during the construction of a multi agent system. The design work is carried out through five models composed by twelve sequential and iterative activities used to produce the MAS specification.

Briefly the models and phases of PASSI are:

1. System Requirements Model. 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 Description, where the system is described in terms of functionalities; (ii) the Agent Identification where agents are introduced and the already identified requirements assigned to them; (iii) the Role Identification where agents' interactions are described by using traditional scenarios; (iv) Task Specification where the operation plan of each agent is draft.

2. Agent Society Model. It is a model of the social interactions and dependencies among the agents of the solution. In the Domain Ontology Description the elements occurring in the system domain are represented in term of concepts, predicates, activities and relationship among them. In the Communication Ontology Description the focus is on the agent's communications that are explained in term of ontology, language and protocol. In the Role Description distinct roles played by agents in the society the tasks involved in each role are detailed.

3. Agent Implementation Model. It is a model of the solution architecture (specific for a FIPA compliant agent platform) in terms of classes and methods required. It is composed of two streams of activities (structure definition and behavior description) both performed at the single-agent and multi-agent level of abstraction.

4. Code Model. It is a model of the solution at the code level. It is largely supported by patterns reuse and automatic code generation.

5. Deployment Model. It is a model of the distribution of the parts of the system across hardware processing unit. Deployment Configuration describes the allocation of agents in the units and any constraint on migration and mobility.

This great number of steps may take a long time to obtain the first prototype code. 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 The Agile PASSI Skeleton

In subsection 2.1 we already represented the needs that arose from our experience in designing robotic systems, we now want to link them to the strategic choices that defines the skeleton (main aspects) of Agile PASSI. The identified requirements and related decisions are:

1. The need of a short process, devoted to the frequent delivery of code; this logically brought us to conceiving an agile process.

2. Another need regards the production of a complete 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, obtaining most of them by some kind of reverse engineering of code. The specific nature of these documents arises from the following points.

3. In order to better deal with the dimension of the systems that 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 agents communications in order to study social relationships and to let the designer to follow the flow of information at run-time. All of these diagrams should provide zero-cost information produced (as already said) by an automatic tool.

4. The opportunity of reusing the growing number of experiences, algorithms and parts of projects, strengthened the role that pattern reuse already played in conventional PASSI[14].

5. Although our systems cannot be considered hard real-time, the opportunity of representing time constraints and concurrent executions has been faced by a behavioral description of the system agents in form of activity diagram (MABD, Multi-Agent Behavior Description).

6. 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, and (ii) the central role of ontology description in describing and analyzing the agent solution.

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

The PASSI methodology starts with a traditional system requirement analysis (**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 **Agent Identification** (AId) activity, use cases of the previous phase are grouped under the responsibility of different agents that assume the responsibility of supplying that functionality. The Agent Identification phase has been maintained in Agile PASSI too, because it allows an early appearance of agents in the process and this grants to start thinking soon *at agent*.

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

No more design oriented fragments are now selected from conventional PASSI. In order to accept the principles of the Agile Manifesto, the next phase facets with the **Code Implementation** for agents of the system. This phase, considering the original PASSI methodology, arrives quite soon in the process, and it is largely supported by a tool (Agent Factory) for automatic compilation of agent structures, patterns reused and automatic code generation. The main features of this tool are:

- Automatic completion from diagrams: the tool analyzes the Agent Identification and Domain Ontology diagram and generates a first skeleton of the agent classes required for the implementation.

- Pattern Reuse: patterns may be introduced in the current project from a repository so enhancing the functionalities of one or more agents 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; this skeleton is completed by methods body coming from the reused patterns. Some experiments have shown a percentage of code reuse that is about 50-60%. Remaining parts of the code have to be added manually by the programmer.

The **Testing** phase plays a fundamental role in all the agile processes because it represents the only way of controlling the correctness of the system and its adherence to requisites. A test suite developed specifically for agent verification completes our development scenario[10]. Test plans are prepared before the coding phase in according with specifications and the AgentFactory tool is also able

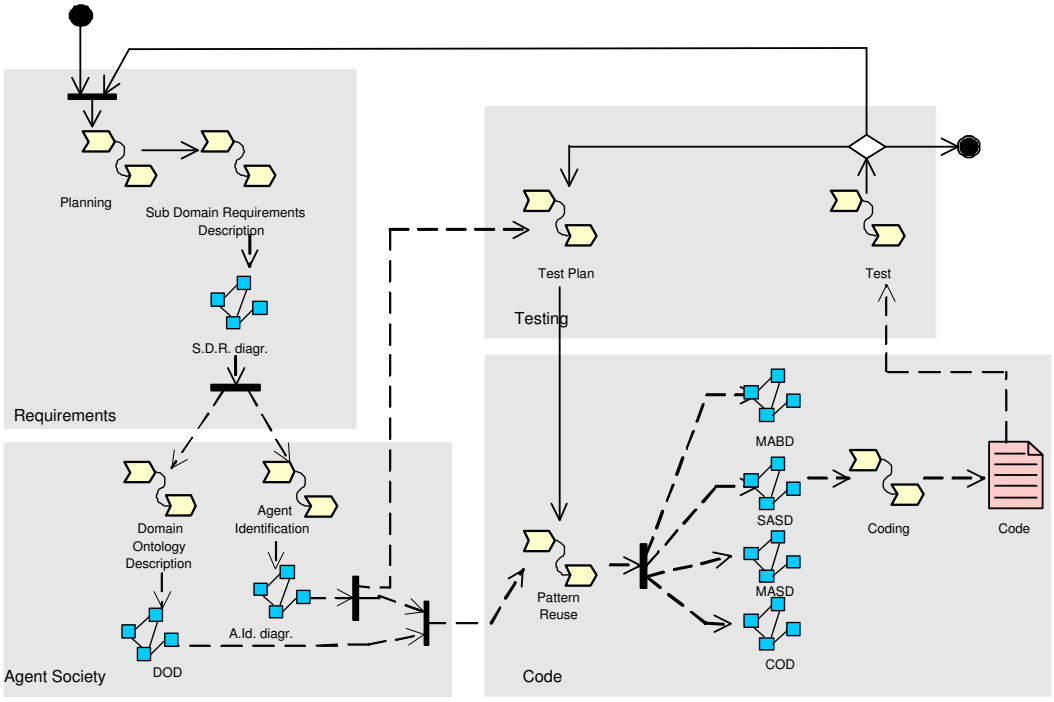

Figure 3. The Agile PASSI process

of generating driver and stub agents for speeding up the test of a specific agent.

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 with a SPEM (the Software Process Engineering Meta-model specification by OMG)[1] activity diagram. There we can distinguish four models:

- 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 code level
- Testing, planned before the code phase and performed soon after it

According to the UML profile proposed by the SPEM specification, in Figure 1 we used three different icons to represent activities to be done in the process (WorkDefinition in SPEM) and artifacts to be produced (UML models or text documents); specifically, a WorkDefinition (like the one used to model "'Planning"') is represented by a couple of hexagons connected with a line, an UML model (like "'Aid. Diagram"') is represented by an icon with four small squares and a text document (like "'Code"') is represented by a typed sheet; the remaining symbols belong to normal UML activity diagrams notation.

4.1 Requirements model

It is composed of two workdefinitions: planning and sub-domain requirements description. In the first one we privilege communication among development team components in order to plan one or more iterations, through a risks and requirements analysis, with the aid of so called user stories (that are typical of XP programming). During this phase the development team decides which activities have to be performed and the order they should be done; the result is a division of the problem in several sub-problems faced in sequential iterations (as prescribed to be in agile methodologies). The resulting iterativity and incrementality are represented in the model by the two main cycles. In the second, common UML use case diagram(s) are used to represent a functional description of the system. The term *sub* refers, as previously said, to the chance of dividing the whole problem in sub-problems.

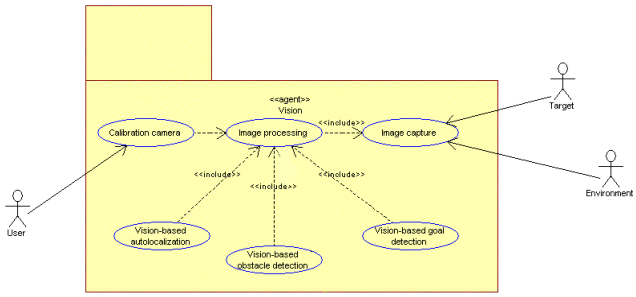

Figure 4. The Agent Identification Diagram designed in the first iteration for reported experiment

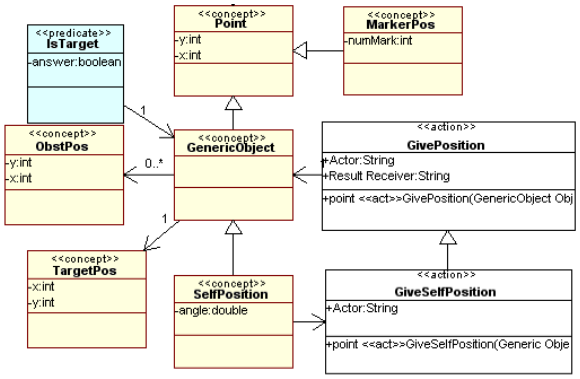

Figure 5. A Domain Ontology Diagram representing a portion of the ontology related to the agent represented in 4

4.2 Agent Society Model

Developing this model involves two work definition: Agent Identification and Domain Ontology description. The first starts from the already produced use case diagrams; according to our definition of agent, it is possible to see an agent as a use case or a package of use cases and starting from a sufficiently detailed diagram of the system functionalities, we group one or more use cases into stereotyped packages so as to form a new diagram, in so doing, each package defines the functionalities of a specific agent; for instance in Figure 4 we can see a portion of A.Id. diagram. Domain ontology description aims to capture the ontology of the system, here involved entities are represented through classes. The ontology is described (using a class diagram, 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 this step is to be done at the same time with the previous one, in fact, while deciding which functionalities will be assigned to different agents we could also identify and represent their knowledge. For instance a Domain Ontology Description diagram reporting some of the concepts is shown in the Figure 5.

4.3 Code Model

This model includes two work definitions: Pattern Reuse and Coding. In the first we try to reuse patterns of agents and we obtain pieces of reusable code that is documented with a structural view and a behavioral one. This is done with aid of a tool that we already adopted in conventional PASSI: Agent Factory; it allows the creation of a multi-agent system by designing new agents or referring to a repository of patterns in order to add them more functionalities; it provides a support for the automatic compilation of a relevant amount of code (not only class skeletons but also inner parts of methods) and it performs the reverse engineering of manually modified code. This pattern-based approach improves project quality, increases the quantity of rapidly produced code and lowers the overall time and costs [23] of development. Since we need a good documentation of the design phase, we specifically produced an add-in for the MetaEdit+ tool that we use to design our systems. This module, starting from the information stored in the Agent Identification diagram and in the structural and behavioral models generated by Agent Factory, automatically produces four documents:

- COD - a class diagram representing agents, their communications and related parameters (content language, agent interaction protocol and referred ontology)
- (M)ASD - 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.
- (M)ABD - an activity diagram representing the flow of control and communications between all the agents.
- SASD - a different class diagram for each agent in order to represent its internal structure and all its task in the most detailed way

In the coding step we complete the code previously produced by putting in practice all the rules of extreme programming.

4.4 Test

The testing phase, in this process, envelopes the coding phase, that is it occurs before and after than coding. This

feature came out from the agile manifesto principles. The agile processes, as the eXtreme Programming (XP), rule that testing must be a continuous activity during the developing process. The testing phase have to start before programming a component (or an agent in this context); in this phase the programmer have to prepare one or more tests that the component must satisfy after the coding phase. This represents a way to take under control the programming work, in fact if almost a test fails the component will be subject to a refinement and a refactoring; this until all the test are satisfied. When the test phase terminates successfully then a working version of the agent is released. This may be not entire according what requisites were included in the test, but it is perfectly running, and it may be used as a prototype to use for a demonstration for the client.

5 Experimental results

The evaluation of a methodology is a complex issue that can be faced from several points of view. In this work we will examine the performance of Agile PASSI in two different ways: first of all, a qualitative exam will be carried on by considering its proximity to some attributes that an agent-oriented methodology should have, then a more quantitative evaluation will be done by repeating some projects (or better significant part of them) already realized with conventional PASSI and comparing the results in terms of development time and support received from the adopted tools. A methodology that is specifically conceived to design MAS, should exhibit some characteristic that are specific of its domain; in this work we will refer to the categories presented by K. Hoa Dam and M. Winikoff in [15]. In this work they compare some agent design methodology by using a questionnarie that has been issued to designers experienced with those approaches and the same methodologies authors. The basic criteria we will consider are:

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

By considering the MAS concepts addressed in Agile PASSI, we can remark that it supports: concurrency (it was one of the requirements identified in subsection 2.1, multi-agent planning (represented in the MABD, Multi-Agent Behavior Description diagram), communications (detailed in all of their most important aspects) and an environment 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 there 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 modeling and notational aspects, Agile PASSI largely refers to UML and extends it only when this is necessary to represent specific issues of agency that could be not coped with an object-oriented notation; again the existing documentation about PASSI (that is even defined according to OMG SPEM specifications [5]) definitely contributes to clarify everything. Obviously the use of our notation will be easy only for designers already skilled with UML and object-oriented design. Traceability is one of the major advantages of Agile PASSI; the add-in that we developed for MetaEdit and the pattern reuse/reverse engineering tool (Agent Factory), give a decisive contribution in this direction, by verifying the consistency among the design artifacts and corresponding code at the 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[22]. 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 supports 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 the field. Nonetheless it is much simpler than the most diffused agent-oriented design methodologies (Gaia, Tropos, Adelfe, Mase an so on) and it is the unique agile one. Almost all the steps of the process are supported by automatisms and tools. Typically these are used to automatically complete/partially compile diagrams by using the information already introduced by the designer/programmer in the previous phases. The programming phase is also well supported in code reuse by the Agent Factory tool. Some limits still exist in the easiness of using all of these tools; they are totally integrated but they could create some problems if some 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 re-

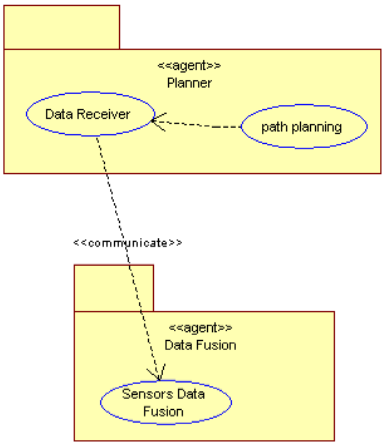

Figure 6. The Planner and DataFusion agents described in section 5

spect to the others). In order to explore more in details the process and to obtain an estimation of the time and the effort required to develop a robotic application with it we built several systems. Now, for space concerns, we will report a simple experiment centered on the navigation of a Koala robot in an unknown environment. The obstacle avoidance is supported by the on board IR sensors and by an eye-bird camera looking at the environment. In Figure 6 a portion of the Agent Identification diagram related to this experiment is reported; the figure shows only a portion of the agents of the system, the Planner and the DataFusion. The Planner agent is responsible for elaborating the best path to the goal using the VFH algorithm[6][19][20]; the DataFusion agent produces a merge of the data collected by different sensors (external camera and on board IR sensors). A portion of the domain ontology is described in Figure 5 and includes concepts like SelfPosition and TargetPosition that are used to plan the robot trajectory. The same system had been developed, in the past, with the conventional PASSI methodology and no one tool or automatic support. The development of this application (without considering the study and tuning of alghoritms) took 2 weeks for the PASSI project and 1 week for the programming and testing phases. The multi agent system produced was composed by seven agents distributed in three computers connected by a LAN. The total dimension of the application is more than five thousands lines of code. Rebuilding the application with Agile PASSI needed only 2 weeks in total; the final release has been obtained after three iterations each of which terminated with a running prototype (the agent involved in the first iteration is described in Figure 4, some others in Figure 6). During the reuse phase, the use of automatism, patterns and the automatic code generation produced a total reuse of approximately of 45-50% of entire code. This represented a significative reduction of the work for the programmer. The documentation obtained from this design experience is good in quality (like can be noted by figures reported in this paper), consistent and largely describes the system and could enable future maintainance.

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 completely fulfills the needs of developing a robotic system. In the last years we adopted the PASSI design methodology and the results were good but, we were recently looking for a new, more versatile and quick process. In building it, we started from the analysis of our requirements, the study of agile processes and method engineering approach; the results of this phases have been conciliated with our existing experience in conventional PASSI by reusing some portions of it in the new process. Agile PASSI is supported by an add-in that we produced for the design tool we adopted (MetaEdit+ by Metacase) and a pattern reuse/reverse engineering application that is a new evolution of the already presented Agent Factory. We already 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 we also describe an example application that has been developed in both conventional PASSI (a couple of years ago) and Agile PASSI (for the purpose of this paper). In the future we will try to enhance the friendliness of the design tools (MetaEdit, our add-in for it and Agent Factory) because their integration is not very transparent to the user and little problems exist in the (automatic) redesign phase of some diagrams whose elements are not correctly re-positioned.

References

[1] Software process engineering metamodel. version 1.0. OMG Document, Nov 2002. http://www.omg.org/technology/documents/formal/spem.htm.

[2] Method fragment definition. FIPA Document, http://www.fipa.org/activities/methodology.html, Nov 2003.

[3] Agile Alliance. *http://www.agilealliance.org*.

[4] M. Cossentino an L. Sabatucci and V. Seidita. Method fragments from the passi process. *Rapporto tecnico ICAR-CNR*, (21-03), 2003.

[5] M. Cossentino an L. Sabatucci and V. Seidita. Spem description of the passi process. *Rapporto tecnico ICAR-CNR*, (20-03), 2003.

[6] J. Borenstein and Y. Koren. The vector field histogram - fast obstacle avoidance for mobile robots. *IEEE Journal of Robotics and Automation*, 7(3):278–288, 1991.

[7] S. Brinkkemper. Method engineering: engineering the information systems development methods and tools. *Information and Software Technology*, 37(11), 1995.

[8] S. Brinkkemper, K. Lyytinen, and R. Welke. Method engineering: Principles of method construction and tool support. *International Federational for Information Processing 65*, 65:336, 1996.

[9] S. Brinkkemper, M. Saeki, and F. Harmsen. Meta-modelling based assembly techniques for situational method engineering. *Information Systems, Vol. 24*, 24, 1999.

[10] G. Caire, M. Cossentino, A. Negri, A. Poggi, and P. Turci. Multi-agent systems implementation and testing. In *Fourth International Symposium: From Agent Theory to Agent Implementation*, Vienna, Austria (EU), April 14-16 2004.

[11] A. Chella, M. Cossentino, R. Pirrone, and A. Ruisi. Modeling ontologies for robotic environments. In *The Fourteenth International Conference on Software Engineering and Knowledge Engineering*, Ischia, ITALY, July 15-19 2002.

[12] M. Cossentino, P. Burrafato, S. Lombardo, and L. Sabatucci. Introducing pattern reuse in the design of multi-agent systems. In *AITA'02 workshop at NODe02*, Erfurt, Germany, 8-9 October 2002.

[13] M. Cossentino and L. Sabatucci. *Agent-Based Manufacturing and Control Systems: New Agile Manufacturing Solutions for Achieving Peak Performance*. CRC Press, April.

[14] M. Cossentino, L. Sabatucci, and A. Chella. A possible approach to the development of robotic multi-agent systems. In *IEEE/WIC IAT'03 Conference*, Halifax - Canada, 13-17 October 2003.

[15] Khanh Hoa Dam and Michael Winikoff. Comparing agent-oriented methodologies. In *Fifth International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2003)*, Melbourne, Australia, 14 July 2003 2003.

[16] Extreme Programming. A gentle introduction. *http://www.extremeprogramming.org*.

[17] Zahia Guessom, Massimo Cossentino, and Juan Pavon. *Methodologies and Software Engineering for Agent Systems*, chapter Roadmap of Agent-Oriented Software Engineering: The European Agentlink Perspective. Kluwer, 2004.

[18] T. Juan, L. Sterling, and Michael Winikoff. Assembling agent oriented software engineering methodologies from features. In *Third International Workshop on Agent-Oriented Software Engineering*, Bologna - Italy, 2002.

[19] J.Ulrich and J.Borenstein. Vfh+: Reliable obstacle avoidance for fast mobile robots. In *IEEE International Conference on Robotics and Automation*, page 1572, Leuven, Belgium, July 15-19.

[20] J.Ulrich and J.Borenstein. Vfh*: Local obstacle avoidance with look-ahead verification. In *IEEE International Conference on Robotics and Automation*, pages 2505–2511, San Francisco (CA), USA, April 2000.

[21] K. Kumar and R.J. Welke. Methodology engineering: a proposal for situation-specific methodology construction. *Challenges and Strategies for Research in Systems Development*, pages 257–269, 1992.

[22] Agile Manifesto. *http://http//agilemanifesto.org*.

[23] M.Cossentino, L.Sabatucci, S.Sorace, and A.Chella. Pattern reuse in the passi methodology. In *ESAW'03*, Imperial College London, UK (EU), 29-31 October 2003.

[24] P. O'Brien and R. Nicol. Fipa - towards a standard for software agents. *BT Technology Journal*, 16(3):51–59, 1998.

[25] J. Ralyt and C. Rolland. An approach for method reengineering. *Lecture Notes in Computer Science*, pages 27–30, 2001.

[26] L. Sabatucci and M. Cossentino. A multi-platform architecture for agent patterns representation and reuse. In *WOA'03 Workshop*, Villasimius (Cagliari) - Italy, 10-11 September 2003.

[27] Motoshi Saeki. Software specification & design methods and method engineering. *International Journal of Software Engineering and Knowledge Engineering*, 1994.

[28] Juha-Pekka Tolvanen. Incremental method engineering with modeling tools: Theoretical principles and empirical evidence (ph.d. thesis). *Jyvskyl Studies in Computer Science*, page 301, 1998.