

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 four 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 [9][1] and extreme programming [6]) 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 [4]; results were interesting[5] 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 an agile version of PASSI. In so doing we took advantage of studies about agent-oriented meta-methodologies[11][10][7] that starting from the method engineering approach born in the object-oriented context [2][8][12], allow the composition of a new methodology by reusing fragments of existing ones and, when necessary, introducing new, specifically created, parts in the process.

2 The Agile PASSI Skeleton

The needs that arose from our experience in designing systems that are deployed on mobile robots lead us to the strategic choices that defines the skeleton (main aspects) of Agile PASSI. 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. We think that all of these issues could be satisfied by using an agile process 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. 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. Our attempt is, now, to reexamine PASSI, using principles and techniques of Agile Methodologies [9][1], 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[6]: (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. 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. In order to accept the principles of the Agile Manifesto, the **Code Implementation** is the most important phase. This, in contrast with 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 func-

tionalties 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 its 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%[5]. 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[3]. Test plans are prepared before the coding phase in according with specifications and the AgentFactory tool is also able of generating driver and stub agents for speeding up the test of a specific agent.

3 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 UML 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

3.1 Requirements model

It is composed of two activities: planning and sub-domain requirements description. During this phase the development team decides which parts of the problem should be faced with in that iteration and lists the work to 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

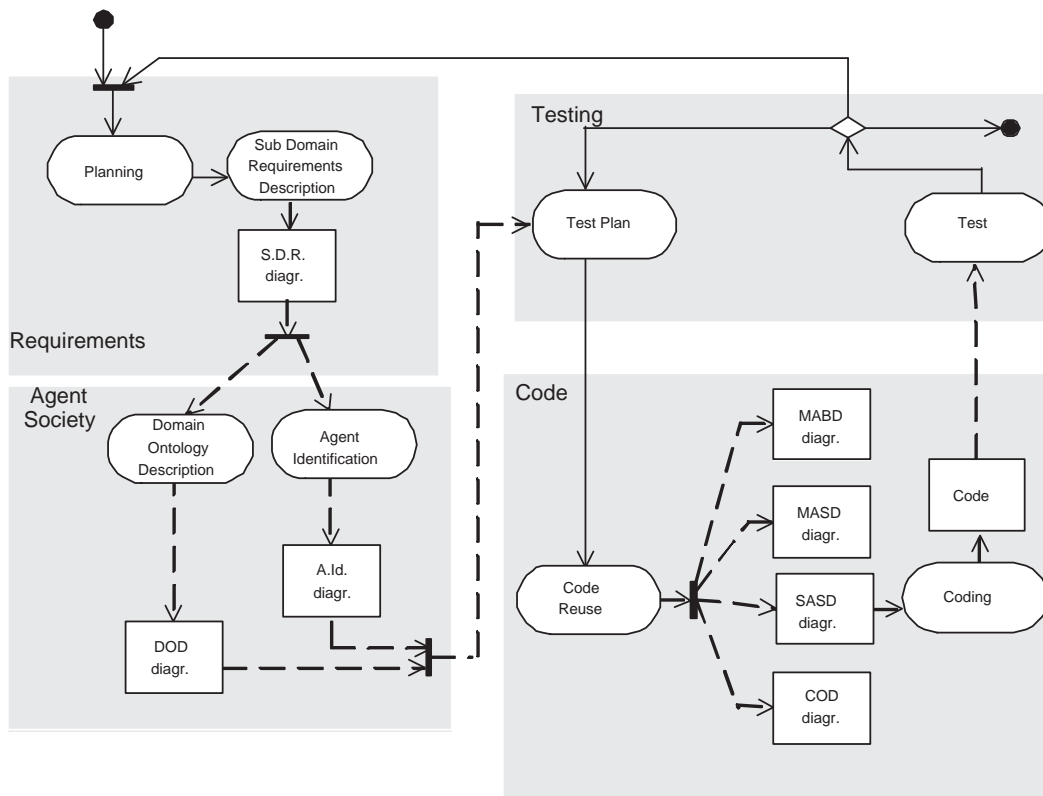


Figure 1. The Agile PASSI process

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.

3.2 Agent Society Model

Developing this model involves two activities: 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 *agent* stereotyped packages so as to form a new diagram, in so doing, each package defines the functionalities that will be under the responsibility of a specific agent. Domain ontology description aims to capture the ontology of the system in terms of concepts, predicates and actions, here involved entities are represented through classes.

3.3 Code Model

This model includes two activities: 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; 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[3].
- (S)ASD - 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.

3.4 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 has to start before programming a component (or an agent in this context); at this stage the programmer has 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 a test fails the component will be subject to a refinement and a refactoring; this until all tests are satisfied. When the test phase terminates successfully then a working version of the agent is released. This may not completely satisfy (yet) all the requirements under its responsibility (new features could be added in another iteration), but it is perfectly running, and it may be used as a prototype for a demonstration to the customer.

4 Conclusions and future works

In this paper we presented a new methodology, Agile PASSI that we conceived in order to have a new approach for 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. 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 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] Agile Alliance. <http://www.agilealliance.org>.
- [2] S. Brinkkemper. Method engineering: engineering the information systems development methods and tools. *Information and Software Technology*, 37(11), 1995.
- [3] 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.
- [4] M. Cossentino and L. Sabatucci. *Agent-Based Manufacturing and Control Systems : New Agile Manufacturing Solutions for Achieving Peak Performance*, chapter Agent System Implementation. CRC Press, 2004.
- [5] 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.
- [6] Extreme Programming. A gentle introduction. <http://www.extremeprogramming.org>.
- [7] 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.
- [8] 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.
- [9] Agile Manifesto. <http://http://agilemanifesto.org>.
- [10] P. O'Brien and R. Nicol. Fipa - towards a standard for software agents. *BT Technology Journal*, 16(3):51–59, 1998.
- [11] 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.
- [12] Motoshi Saeki. Software specification & design methods and method engineering. *International Journal of Software Engineering and Knowledge Engineering*, 1994.