Agile PASSI

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

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

June 22, 2004

1 The Agile PASSI Skeleton

In the past, we developed a large amount of systems by using PASSI, results were interesting[4] 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 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 (Agile PASSI [3]). In this work, 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. 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 analyzing the agent solution.

Our choices have been also conditioned by some of the fundamental strategies of the Agile Manifesto[5]: (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. All of these arguments brought us to identify the parts of PASSI 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, one of the most important phase is the code production; 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 reuse, that is a new evolution of the already presented Agent Factory, 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[2]. 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.

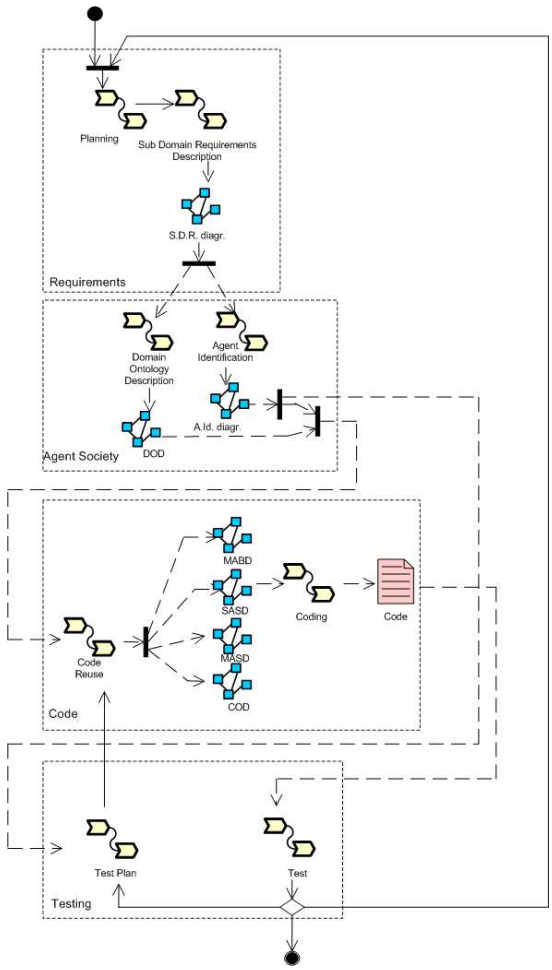

Figure 1: The Agile PASSI process

2 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 [3] 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.

2.1 Requirements model

It is composed of two workdefinitions: planning and sub-domain requirements description. During the first 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.

2.2 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 we can see a portion of A.Id. diagram.

2.3 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[6]. 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.

2.4 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.

References

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

[2] 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.

[3] A. Chella, M. Cossentino, L. Sabatucci, and V. Seidita. From passi to agile passi: tailoring a design process to meet new needs. In *IEEE/WIC/ACM*

Conference on Intelligent Agent Technology (IAT 2004), Beijing - China, 20-24 September 2004.

[4] 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.

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

[6] 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.