Composition of a New Process to Meet Agile Needs Using Method Engineering

Massimo Cossentino, 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, seidita@pa.icar.cnr.it

Abstract

The need of developing a new software engineering process (SEP) that could allow the quick prototyping of some robotic applications and meet the requests coming from some companies for a development process that was shorter than PASSI, gave us the opportunity of applying our studies on the assembling of a new SEP by reusing parts (called method fragments) from other processes. In this paper we discuss our approach that, starting from the method engineering paradigm, adapts and extends it considering specific agent-oriented issues like the multi-agent system metamodel. The final result of our experiment (Agile PASSI) is presented together with the requirements that motivated its structure.

1 Introduction

Many different design methodologies for multi-agent systems could already be found in literature and nonetheless further works propose brand new approaches or the extensions of existing ones. We think (but the opinion is largely shared in the scientific community) this happens because each methodology has been conceived to solve a specific problem in a fixed context and this strongly limits the possibility of reusing it (without significant changes) in a different situation. Several developers respond to their need of designing a specific system in some productive context by creating a specific design methodology; this implies a big effort and the cost of developing a MAS (multi-agent system) becomes higher than the comparable object-oriented solution (it is worth to note that in the object-oriented context the Unified Process is an accepted standard and designers does not need to add the design process construction cost to the development effort).

A new branch of Software Engineering, called Method Engineering [1, 2] proposes of creating a new methodology starting from existing methodology parts, called *method fragments*, that a method engineer defines and stores in the method base. When he wants to design a new methodology, he extracts and assembles the fragments (each one composed of some work to be done, the resulting artifacts and supporting guideline) in order to obtain a methodology that is suitable for his specific needs. Because of the great number of methodologies that could be used to extract method fragments, it is necessary to represent them in a standard way and to have a definition of the method fragments that could fit it. This work consists in a re-engineering process [3] of existing methodologies to identify and extract fragments that could be used in the new methodology construction process. We think that in the AOSE (Agent-Oriented Software Engineering) context, some confusion still exists among the use of the terms process, methodology and method. In order to avoid misunderstandings, in this work, from now on, we decided to refer to the (design) process (avoiding the use of the word methodology) meaning with it the collection of phases, activities and steps that produce the project deliverables. The term method will be used with the meaning of a way of performing some kind of activity (at whatever level) within the design process (this includes techniques, artifacts, and guidelines).

As a consequence of this adopted terminology, we will refer to the final result of the method engineering activity as a new *process* or indifferently SEP (Software Engineering Process). It will be composed by a set of method fragments, each one of them specifying which phase/activities or more generally work definitions should be carried on and by which stakeholders. The most frequent aim of these work definitions is producing/refining one or more artifacts (text documents, diagrams, ...) and in so doing they often refer to some kind of style template (text documents) or modeling language (diagrams). This process in order to be successfully applicable should be complemented by some guidelines that will help the involved stakeholders in performing their duties according to some defined best practices. The process will also prescribe in which sequence the phases and activities will be executed and if iterations

should be done or feedbacks provided to previous items; this often relates to some common models like the waterfall [4] and evolutionary [5] (including iterative and incremental) ones.

As already said, before proceeding to fragments assembling, we need to describe and represent these parts in a standard way so to make easy the composition of parts coming from different processes. The first step of this work consists in the creation of the meta-model that will be used to describe the existing processes and the multi-agent system structure. An important contribution to the solution of the first issue comes from an OMG specification, the Software Process Engineering Metamodel [6]; this is the natural candidate to become the adopted process meta-model, since it is already an accepted standard in the OO context. We have exploited the possibilities offered by SPEM in the specific agent-oriented context obtaining interesting results in the modular representation of PASSI [7] and the method fragment extraction from it. In this paper we will present our approach to the reuse of these fragments for assembling a new process (Agile PASSI) that satisfies our specific robotic applications development needs.

The paper is organized as follows: in the following section we present an introduction to the key topics of this work: method engineering and agile processes; in section 3 we present our general approach to the new process composition and related method fragments selection; in section 4 we quickly present the PASSI process from which we extracted the method fragments; in section 5 we report the results of our experiment, and finally 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 an agent-oriented application accepting low compromises on the quality of the design and its documentation. This brought us to identify the need for an agile process that could be supported by some design tool. Taking profit of our previous experience with the PASSI process [8], patterns reuse [9], and related design tools [10], 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 2.1 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 Agent-Oriented Method Engineering

In order to build our new design process we adopted (and extended) the method engineering paradigm [11][12][13]. According to this approach, the new SEP (Software Engineering Process) is built by assembling pieces of the process (method fragments) [2][1][3] 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 [14], and is now collecting a growing interest from the agent community[15][16].

Some differences exist between the approach we used in building Agile PASSI and the cited approaches in the object-oriented context; the most relevant one is that in the OO context the construction of method fragments, the assembling of the new SEP 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 (meta-)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 role, are used, by different authors and in different processes, with slightly distinct meanings or granularity. We built Agile PASSI by adopting the MAS meta-model represented in Figure 2, that will presented more in details in sub-section 3.1.

Before introducing the process we adopted to create Agile PASSI, it is worth to provide a definition of method fragment. In this work we consider a **method fragment** (or briefly a fragment) as composed by:

1. A portion of process;
2. One or more deliverables (artifacts like (A)UML/UML diagrams, text documents and so on);
3. Some preconditions (like required input data or guard conditions);
4. A list of concepts (related to the MAS meta-model) to be defined/designed/refined by executing the specific method fragment;
5. Guideline(s) that illustrates how to apply the fragment and best practices related to that;
6. A glossary of terms used in the fragment;

7. Other information (composition guidelines, platform to be used, application area and dependency relationships useful to assemble fragments) complete this definition.

In Figure 1 it is presented what we think to be the correct process for composing a new SEP under the evolution of the method engineering paradigm that we call agent-oriented method engineering. The process begins with the introduction in the method base of the fragments extracted from available processes and the specifically created new ones; then the designer (or better the method engineer), before building the new SEP, identifies the elements composing the meta-model of the kind of MAS he wants to build. The composition of the new SEP is performed under the assistance of some specific software tool, called CAPE (Computer Aided Process Engineering) or CAME (Computer Aided Method Engineering) depending on its process or method-oriented vocation. This tool will allow the selection of the right method fragments from the method base and will permit their introduction in the selected (or specifically designed) process model.

In this process, the definition of the MAS meta-model will help at both a logical and practical level. Firstly this will be useful in the method fragment selection phase (avoiding the selection of methods dealing with elements that are not present in the defined MAS meta-model) and secondly, the same fact of clearly declaring the structure of the system will allow the design tool to check for model coherence and to find not completely defined parts. Once the new SEP has been composed, the same CAPE/CAME tool should permit the instantiation of a simpler tool (a CASE, Computer Aided Software Engineering, tool) that will be used by the designer when designing a system to solve some specific problem.

Agile PASSI has been constructed according to this process and in defining/composing our fragments we used a CAME tool (MetaEdit+ by Metacase) that offered a specific support for the composition of a process from existing fragments.

2.2 Agile Processes

Classic SEP 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 a software,in the last years a new kind of processes, called lightweight in a first time but now known as agile, has been developed. An important difference between the two kinds of processes is the smaller quantity of documentation produced in the second case, in fact agile ones are code-oriented being source code the key element of documentation. The large quantity of high level documentation we create while performing a classic SEP induces some limitations in facing changes; a very powerful way to take under control continuous changes is modeling through little increments, producing working portions 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 processes (often called agile methodologies) are not complete processes 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 processes [17], in order to create a lightweight SEP, simple, easy to use and principally based on code production rather than on documentation (that is still requested, but mostly 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 approaches, Extreme Programming [18]: (i) Planning, (ii) Designing, (iii) Coding, and (iv) Testing. As it will be presented later, this sequence will constitute the center of the proposed SEP.

3 The proposed approach

This section proposes our approach to the composition of a new process. In this specific work we will apply our ideas to the reuse of PASSI fragments in order to build a new process (Agile PASSI) accordingly to some requirements that will be presented later. In our research activity we decided to adopt existing standards whenever possible in order to remain as close as possible to industrial needs in this direction; for this reason we adopt: SPEM (Software Process Engineering Metamodel) by OMG [6] in modeling our processes (and related fragments), UML (extending it when necessary) in modeling our artifacts; FIPA [19] as the reference agent architecture and XML for data representation.

In creating a new process, we consider that this is essentially a design activity by itself and as such it should be ruled by some kind of design process. The design process we adopt (to design a new process) is composed of four phases: requirements analysis, process model design, fragments selection, and fragments integration (it includes the assembly and adjustment activities performed to adapt the fragments to the new process). Further iterations in this

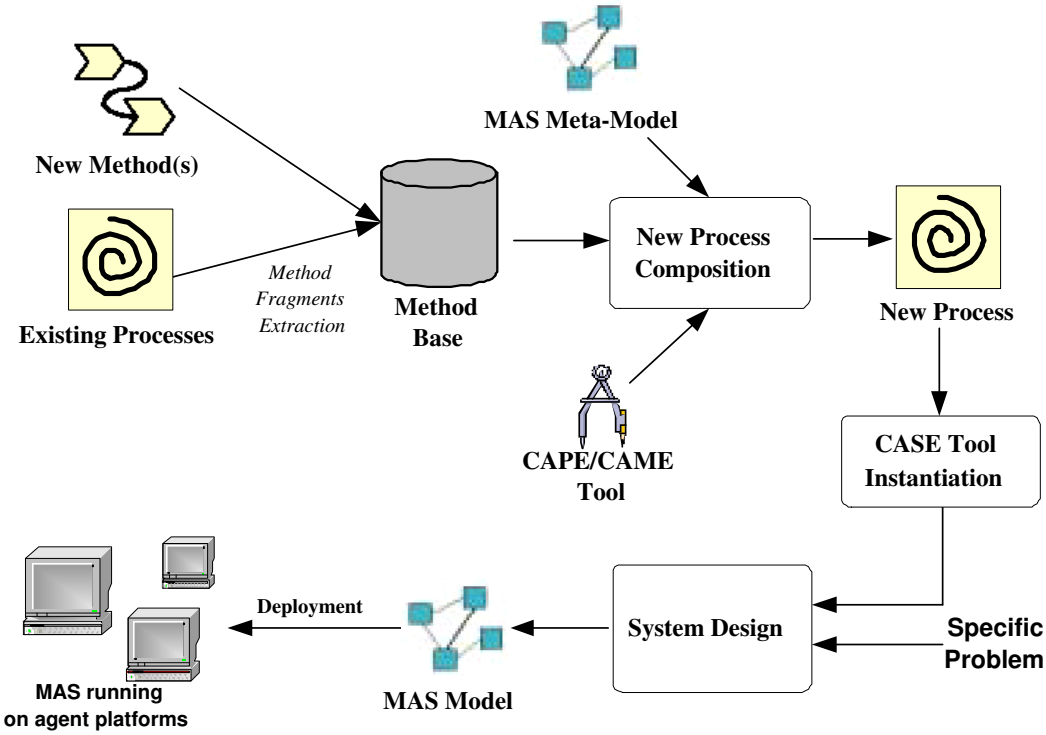

Figure 1. The adopted Agent-Oriented Method Engineering process

sequence of phases should aim at process maturity as described in the CMM [20] but these aspects are out of the scope of this paper.

The **Requirements Analysis** phase, consists in the identification of the important features of the process under construction; for instance the need of an highly detailed level of design that could derive from a defense contract project. Another example could be the indisputable dependability requested to a mission critical system like an avionic one. The **Process Model Design** consists in the selection of the process model (waterfall [4], evolutionary or incremental [5], transformation [21], spiral [22], ...), the phases that constitute it and other process level requisite (for instance the conditions that enable each new iteration). We consider situational requirements as the most useful guidance for selecting the right process model. The need of facing with rapidly changing requirements could bring to the adoption of an evolutionary process while, conversely the need of a very formal development process, with high quality level insurance could lead to the adoption of some IEEE guidance [4][23] and therefore to the selection of a waterfall model. The **Fragments Selection** phase aims at identifing the best fragments for achieving the process goals (according to the requirements identified in the first phase). Some authors (Ralyté et al. [2]) identify method fragments (called 'chunk' in that work) using a process-driven very structured and complete heuristic. We think that this top-down method, although very clear and well defined is not sufficient to meet all the requirements (that are often expressed also in terms of deliverables and architecture of the system to be developed). For this reason we found useful to complement a process driven selection activity with another data-driven one that considers aspects like diagrams/other documents to be produced and the system architecture according to some kind of MAS (multi-agent system) meta-model. From the process-driven point of view, we consider four different levels of method fragment granularity according to the position of the fragment in the process (in this classification we adopt the SPEM terminology): *Phase* (highest level parts of the process, usually characterized by an entry condition, a goal and the sequentiality constraint, for instance System Requirements and Agent Society in PASSI), *Work Definition* (a substantial part of the operations to be performed in the process, usually it is composed of several lower level elements; for instance Agent Identification, and Domain Ontology Description in PASSI, see section 4), *Activity* (usually the smallest reusable part of a process, an activity is composed by the tasks, operations, and actions that are performed by a role or with which the role may assist, for instance Use Case Identification and Roles Dependencies Analysis in PASSI [7]), *Step* (the atomic elements that compose an activity, for instance the different steps of

the heuristic used for identifying agents from use cases in PASSI)

The selection of method fragments (that in our approach could be at the *Phase*, *Work Definition* or *Activity* level of granularity) is performed working on two dimensions: the process dimension and the system architecture dimension. The *process dimension* enables a zooming on the analysis done during the process model design and considers lower level features of the process. For instance at this stage we evaluate the need for a specific attention on security (from which we will deduce the importance of introducing some specific method fragment). Essentially in this phase we first select the phases we want to introduce in the process (while some process model like the waterfall one already prescribe these phases, some others leave a considerable level of degree in this choice), and then we select the lower level fragments inside them. In so doing we follow some criteria:

- Process completeness: all phases (and their activities) of the process are to be covered by appropriate method fragments;
- Process coherence: generally speaking, each fragment refers to some kind of 'philosophical' or 'methodological' approach to the solution of the problem it faces. It makes no sense to introduce fragments belonging to contrasting approaches in the same process;
- Process applicability: the selected fragments should compose a process that is realistic (not too complex or simplistic for the faced problem) and lead to the final solution in an optimal (or at least acceptable) way (in terms of cost and time);
- Contracts accomplishing: each fragment has some specific preconditions that should be enacted by previous parts of the process and when it has been applied, it generates some postconditions that could trigger the following fragments;
- Stakeholders adequacy: it consists in selecting a set of fragments where the skills required for involved roles (analyst, architect, programmer, ...) are adequate to the situation (company, developing team, ...) where the process will be applied;
- Stakeholders satisfaction: people involved in the process application play a decisive role in the success of the project. Their expectancy in terms of the kind of work they will participate, is an important factor for the selection of fragments.
- Specific requirements: they could help in the selection of some fragments. For instance the need of designing a real-time system will induce to consider fragments that deal with time-related aspects of the design.

In the *system architecture dimension* we define the MAS meta-model and from it we deduce the need for specific fragments that with their resulting artifacts could contribute to the definition of a system obeying to the defined meta-model. We now deduce the models and views that are necessary to define and refine the elements of the system (this in someway resembles the product perspective of Brinkkemper et al. in [3]).

During the **Fragments Integration** phase, the selected fragments are disposed in the right position inside the process and when necessary they are adapted to the new context. Method fragment contracts (preconditions required by each fragment and postconditions enacted by it) are used to verify the possibility of directly connecting some fragments. An interesting approach to the adaptation of fragments is described in [24].

3.1 The Agile PASSI process composition experiment

The reported experiment started from two different motivations, the first was that we needed a short design process to let designers focus on the implementation of relatively small robotic applications; the second motivation was that during the development of large projects some of ours industrial partners underlined the benefit that could come from the availability of a versatile process that could substitute PASSI in the development of minor parts of the whole project. Because of space concerns, in the following we will only refer to the first motivation but the other has been considered too during the Agile PASSI construction and the resulting process proved good in the developing of non robotics applications too. Our robotic 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). We now want to design a process that, taking profit of the successful experience already done with PASSI, could be the best solution for this kind of problems in our laboratory context.

The requirements that we could identify for our new process are centered on the main goal of not distracting developers from their main objective of implementing/tuning some kind of new algorithm with a long design process; nevertheless, we do still need to maintain a reasonable quality of design documentation for enabling the knowledge transfer among people in our laboratory. 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. The last concern is about the design of the real-time aspects of the application. Although our

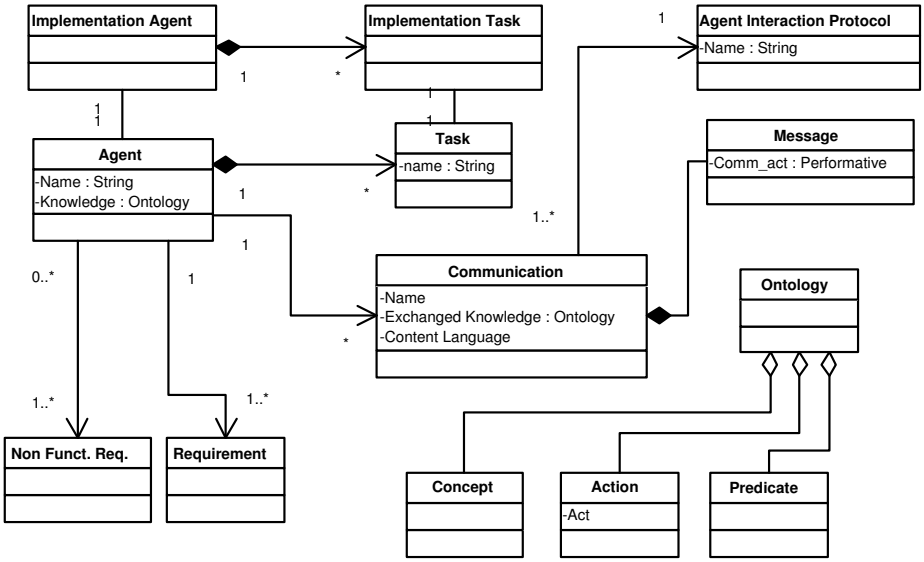

Figure 2. The Agile PASSI MAS meta-model

robots move slow, the use of low efficiency agent platforms (Java-based) could bring to an unacceptable decay in performance if no specific attention is given to this problem.

As regards the other dimension we consider in our composition approach (the system architecture), the requirements of the new process regard our decision of significantly reducing the dimension of the conventional PASSI MAS meta-model [25] because of the direct relationship that exists between the number of elements of the meta-model and the design artifacts (and activities). The chosen MAS meta-model is reported in Figure 2, it is composed of four different categories of elements: requirements (functional and non functional requirements), domain ontology (concept, predicate, action), agent logical (abstract) structure (agent, task, communication, message, agent interaction protocol), and agent implementation structure (implementation agent and implementation task).

In this meta-model, 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 terms 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)

From all of these requirements we deducted some choices for our new process:

- we decided to adopt an agile process. This introduces a specific structure of process model: (a) it should be short (composed of only a few phases), iterative, and incremental (as a consequence we need some iteration planning activities) and (b) a specific attention is devoted to coding and testing in order to have a frequent delivery of functional portions of the final system; this solves the developer 'anxiety' of focusing on algorithm implementation rather than system design.

- The process should be composed of a quick design phase and should encourage the reuse of portions of existing design artifacts and applications in form of patterns; it should enable the automatic production of a consistent documentation at different levels of abstraction by re-engineering the produced code.

- The design aspects we decided to maintain from conventional PASSI are related to the initial part of the process (use case based requirements analysis) and the agent society model (functionality-based agents identification and a detailed domain ontology design). This satisfies the expectancy of already skilled PASSI designers that do not want to study a totally new process.

- Finally, the process has to be supported by a specifically conceived design tool in order to limit all the

operations that are performed 'by hand' (this also includes design documentation production) because they contribute in significantly slowing down the process and could introduce mistakes in the final result.

The resulting process is reported in Figure 3 and it is composed of five different phases: (i) *Requirements* where the new iteration is planned (in terms of risks and requirements to be faced) and a use case based analysis of system requisites is performed; (ii) *Agent Society* where the agents that will constitute the system are identified and the domain application ontology defined; (iii) *Test Plan* where starting from requirements, a detailed plan of the test that will be applied to the code is prepared; (iv) *Coding* where code is produced (with patterns reuse); and (v) *Testing* where the produced portion of the system is tested accordingly to the previously prepared test plan.

4 PASSI Description

PASSI [8] is a process for multi agent systems development that covers all the design activities from the requirements analysis to the system implementation and deployment. The design work is carried out adopting five phases composed by twelve sequential and iterative work definitions used to produce the MAS specification.

Briefly the phases and work definitions of PASSI (in Figure 4 a SPEM diagram representing them) are:

1. **System Requirements**. It is composed of four different work definitions and produces a description of the functionalities required for the system and an initial decomposition of them accordingly to the agent paradigm. The four work definitions are: (i) the *Domain (Requirements) 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) the *Task Specification* where the operational plan of each agent is draft.

2. **Agent Society**. It composes a model of the social interactions and dependencies among the agents of the solution. It is composed of four work definitions: in the *Domain Ontology Description* the elements occurring in the system domain are represented in terms of concepts, predicates, actions and relationships among them; in the *Communication Ontology Description* the focus is on agent's communications that are explained in terms of referred ontology, content language and agent interaction protocol; in the *Role Description* distinct roles played by agents in the society and the involved tasks/behaviors are detailed; in the *Protocol Definition* non-standard agent interaction protocols are defined.

3. **Agent Implementation**. It is a model of the solution architecture in terms of classes and methods required. It is composed of four work definitions organized in two streams of activities (structure definition and behavior 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* work definition, describes the allocation of agents in the units and any constraint on migration and mobility.

Testing in PASSI is divided in two different stages: the *Agent Test* where each single agent is tested after its implementation (Code phase) and the *Society Test* where the whole multi-agent system is tested (after the Deployment phase).

This great number of steps may take a long time to obtain the first prototype code. Also, the process is iterative both among the phases and in the whole life cycle; this configures PASSI as a traditional process in which the coding phase is positioned somehow late in the process and like many other classical approaches it is oriented to high level documentation production, and it is more adequate for projects with a low level of changes in requirements.

From PASSI we extracted several fragments some of which will be reused or adapted for the creation of the Agile PASSI process. In the following subsection we will describe the PASSI method fragments extraction process.

4.1 PASSI fragments

Before performing the fragments extraction from PASSI, we re-engineered it in order to represent all the process aspects (activities, artifacts, constraints and conditions) in a way that could enable the method fragments identification. SPEM (Software Process Engineering Metamodel [6]) was adopted as a process meta-modeling language; this language allows an intuitive description of the software development process and its components and includes an UML profile that can be used to graphically represent the process using UML activity, class and use case diagrams. The core of SPEM is in its conceptual model: a software development process can be seen as a collaboration between abstract active entities called *Process Roles* that perform some operations called *Activities* on concrete entities called *Work Products*.

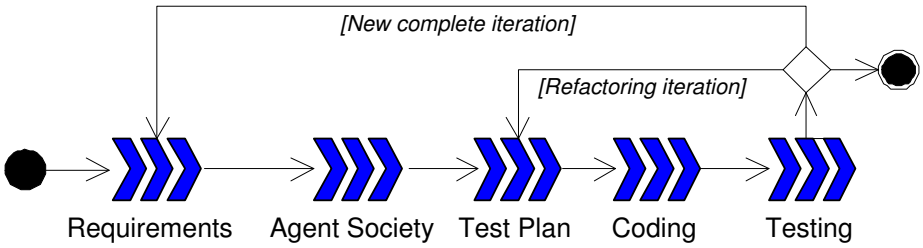

Figure 3. The phases of the Agile PASSI process

We represented the PASSI process in SPEM using two sequential steps, in the first we considered the whole process with the involved disciplines, in the second we detailed the separate phases and work definitions, following the conceptual model described above and the method fragment structure already defined in section 3. Starting from the procedural representation of PASSI composed of five phases (Figure 4), we decided to extract one different fragment for each one of the PASSI work definitions (refer to the beginning of this section for their list). In so doing we obtained a substantial simplification of our new process creation work: the assembly process will only deal with two levels of fragments (phase and work definitions); another consequence is that the modifications of them during fragments integration will be easier since it will mainly deal with work definition level fragments (and rarely with their composing activities).

At the end of our PASSI re-engineering and fragments extraction work we obtained seventeen work definition level fragments and five phase level ones; they constitute the fragments repository from which we selected the elements for composing the Agile PASSI process

5 The Resulting Agile PASSI Process

Starting for the considerations proposed in the previous sections we selected from the PASSI process some fragments that we consider in line with the new process requirements (see subsection 3.1) and our 'philosophy' in agents development (use case based agents identification and central role of ontology); the selected method fragments: Domain Requirements Description (a description of the system requirements in terms of use cases), Agent Identification (the clustering of system functionalities into packages associated to agents), Domain Ontology Description (an ontological description of the solution domain in terms of concepts, predicates and actions), Code reuse (a comprehensive pattern reuse technique that allows the automatic production of code) and Testing (of agents and societies). The new Agile Process (reported in Figure 5 in form of a SPEM activity diagram), resulting from the composition of these work definitions in the five phases of the general model proposed in Figure 3, it is composed of eight work definitions (*Planning*, *Sub-Domain Requirements Description*, *Domain Ontology Description*, *Agent Identification*, *Pattern Reuse*, *Coding*, *Test Plan*, *Test*) and eleven artifacts (seven UML diagrams and four text documents).

More in details, the first phase (**Requirements**) consists in an high level analysis of the system under construction through two sequential work definitions:

- *Planning*, where through the communication among team elements and sequential iterations the problem is divided into sub-problems so to make possible a correct risks management and activities scheduling. This first activity should result in a text document (*Iteration Plan*) summarizing the considerations and the solution proposed by team elements.
- *Sub Domain Requirements Description*, a functional description of the system through common UML use case diagrams. This work definition corresponds to the PASSI *Domain requirements Description*, the 'Sub Domain' prefix has been added to stress the incremental concepts that are behind this process.

In the **Agent Society** phase, developer identifies the agents involved in the solution (assigning the previously identified functionalities to them), and then he defines the ontology of the domain. The phase is composed of two parts:

- *Agent Identification*, in this activity, starting from the previously produced use case diagram, another one is composed clustering use cases in packages that represent the functionalities assigned to agents; in this way, each agent will be responsible for the satisfaction of some requisites.
- *Domain Ontology Description*, the domain is expressed in terms of its ontology through a class diagram where classes represent concepts, predicates and actions.

We expect that this two work definitions are carried on iteratively; after the identification of an agent the definition of its knowledge and actions starts and this could bring to some changes in the list of functionalities assigned to it.

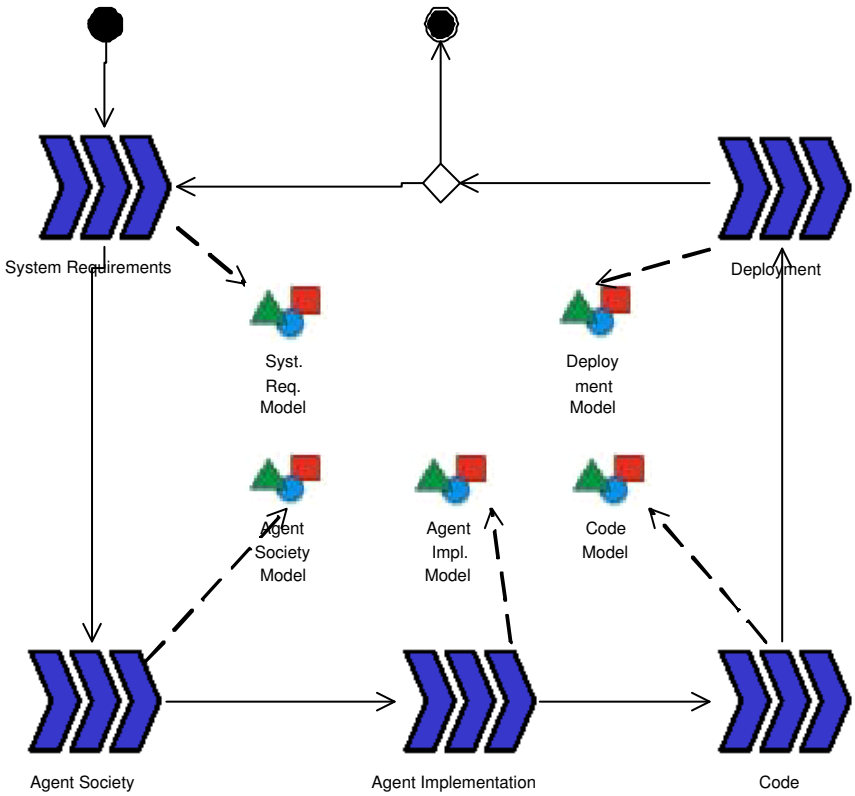

Figure 4. The phases of the PASSI process

Testing is a continuous activity during an agile development process, in Agile PASSI this is divided in two phases; the first is **Test Plan**, that has been conceived referring to the agile processes principles and particularly to eXtreme Programming [18] rules; according to these rules the testing phase starts before the coding activity, the designer/programmer has to first prepare the test plans and then coding the component that must satisfy them (this will be proved during the following *Testing* phase).

The **Code** phase is composed of two strictly coupled parts.

- *Patterns reuse*, where we try to reuse portions of precedent projects through the reuse of patterns of services (interactions among agents), agents, tasks and actions. In this activity the Agent Factory tool proves very useful allowing us the automatic generation of relevant portions of code and a reduction of development time and costs.
- *Coding*, consists in the introduction of the code that cannot be derived from patterns (for instance problem specific algorithms).

Coding phase is the core of Agile PASSI and it is largely supported by a tool, APTK (Agile PASSI Toolkit), that is an add-in of a commercial design tool (Metaedit+). APTK offers several features to the designer, its main functionalities are :

- Automatic compilation of diagrams - this allows the partial drawing of some diagrams, for instance the Agent Identification diagram is initially drawn reporting the use cases of the previous work definition, and the complete design of some others starting from the code re-engineering and other design information (like applied patterns), for instance the Communication Ontology diagram is composed in this way.
- Support of changes - our tool, interacting with the Metaedit+ functionalities, allows the user to modify all the design models (even those automatically generated by the tool), and to profitably perform an incremental and iterative development of the project.
- Consistency check - the developer can perform a check on all the generated models to verify their consistency or he can use the MetaEdit+ checking feature for verifying the correctness and consistency of each single diagram.
- Report and project documentation generation - APTK

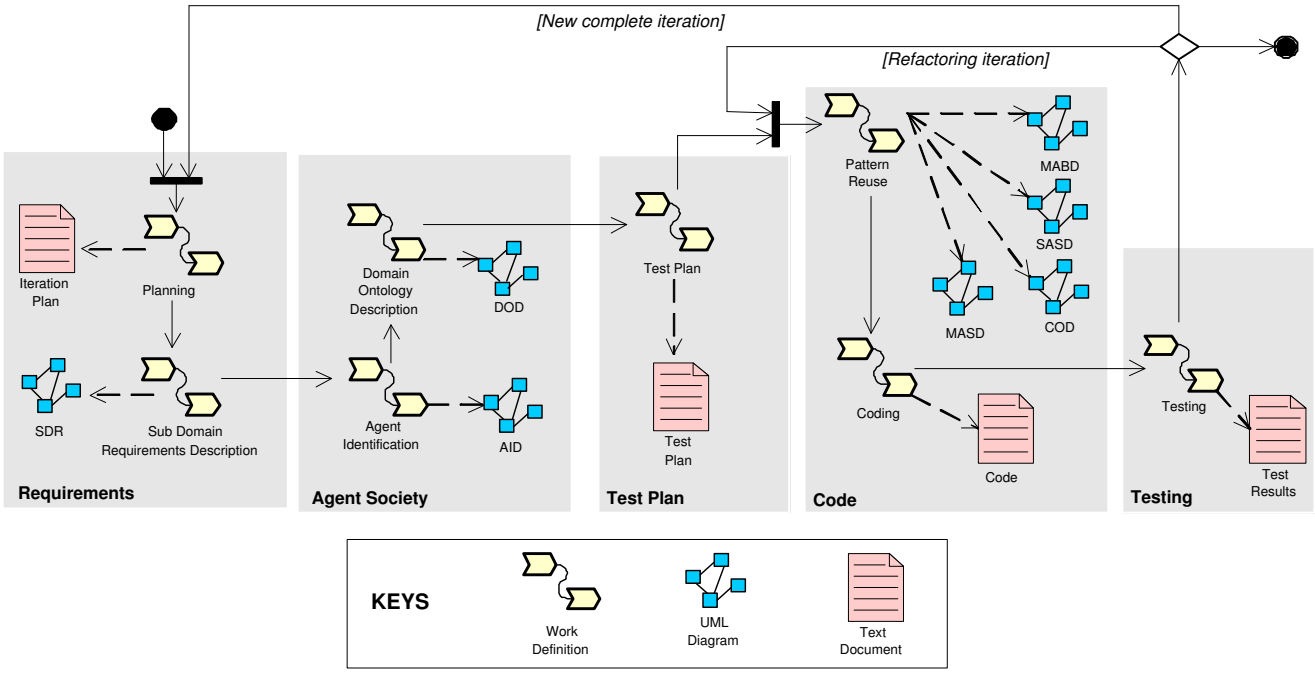

Figure 5. The Agile PASSI process

allows the creation of MS Word or HTML documents representing all the design aspects.

- Patterns reuse - the user interacts with Agent Factory, that is totally integrated with APTK, to apply patterns to the system and generate their code.
- Automatic code generation and reverse engineering - Code generation and reverse engineering are entirely done by the Agent Factory application, through its integration in APTK.

Testing, after code completion, is the phase where the real test accordingly to the previously defined test plans is performed.

Agile PASSI has been created starting from conventional PASSI with the precise aim of having a lighter design process that could fit the needs arising from the development of small-medium size projects. As a consequence there are not fundamental differences between the two processes with the exception of those that we can individuate between a classic SEP and an agile one. Even one of the most agent-oriented aspects of a design process (the MAS meta-model) is not very different. In building Agile PASSI we referred to the MAS meta-model represented in Figure 2 whose elements are a subset of the conventional PASSI MAS meta-model [25].

Being our process agile, it is iterative, composed by a low number of steps and it strongly involves the end-user (or customer) during the development phases. These are choices we did in order to be compliant with the agile manifest principles[17], and as a consequence some of the phases of traditional SEPs are not considered (this is the case of the system architecture design that is left to the agent society organization) or performed very quickly. Quality assurance is enhanced by the large reuse of patterns, the automatic production of relevant portions of code and the consistency check performed by the tool on the design artifacts.

6 Conclusions and Future Works

This work started from the need of developing a new software engineering process (SEP) that could allow the quick prototyping of agent-oriented applications. In previous experiences we used the PASSI process that proved good for the development of medium-large size applications but it was too time consuming for the development of smaller size applications. This gave us the opportunity of applying our studies on the assembling of a new SEP by reusing parts (called method fragments) from other processes. This approach already known as method engineering in the object-oriented context it is now diffusing in the agent-oriented community as a logical attempt of rationalizing and reusing the great number of development processes proposed in literature.

In this paper we discuss our approach that is composed of four phases: (new SEP) *Requirements Analysis, Process Model Design, Fragments Selection, Fragments Inte-*

gration. In these phases we also consider specific agency peculiarities like the MAS meta-model that differently from what happens in the object-oriented context is not a-priori known and fixed, but it is one of the most important differences that can be found in the development processes proposed in literature. The final result of this work (the Agile PASSI process) is finally presented starting from the requirements that motivated its structure.

In the future we aim at further detailing the different aspects of our approach, by formalizing a sufficient number of techniques and guidelines that could efficiently support the work of the method engineer. As regards the Agile PASSI process, after having applied it in a couple of small projects, we can say that it fully achieved the goals we were pursuing from the methodological point of view while the design tool (APTK) has still to be significantly improved in order to reach the flexibility and extensive support offered by the conventional PASSI support tool (PTK).

References

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

[2] Ralyte, J., Rolland, C.: An approach for method reengineering. Lecture Notes in Computer Science (2001) 27–30

[3] Brinkkemper, S., Saeki, M., Harmsen, F.: Meta-modelling based assembly techniques for situational method engineering. Information Systems **24** (1999)

[4] Board, I.S.: Ieee std 1074-1997, standard for developing software life cycle processes (1997)

[5] Gilb, T.: Principles of Software Engineering Management. Addison-Wesley Reading (1988)

[6] OMG: Software Process Engineering Metamodel Specification. http://www.omg.org (2002)

[7] Cossentino, M., Sabatucci, L., Seidita, V.: Spem description of the passi process. Technical Report 20-03, ICAR-CNR (2003) Available online at http://www.pa.icar.cnr.it/ cossentino/ FIPAmeth/metamodel.htm.

[8] Cossentino, M., Sabatucci, L.: Agent system implementation. In Paolucci, M., Sacile, R., eds.: Agent-Based Manufacturing and Control Systems: New Agile Manufacturing Solutions for Achieving Peak Performance, CRC Press (2004)

[9] Cossentino, M., Sabatucci, L., Chella, A.: A possible approach to the development of robotic multi-agent systems. In: IEEE/WIC IAT'03 Conference, Halifax - Canada (2003)

[10] M.Cossentino, L.Sabatucci, S.Sorace, A.Chella: Pattern reuse in the passi methodology. In: ESAW'03, Imperial College London, UK (EU) (2003)

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

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

[13] Saeki, M.: Software specification & design methods and method engineering. International Journal of Software Engineering and Knowledge Engineering (1994)

[14] Tolvanen, J.P.: Incremental method engineering with modeling tools: Theoretical principles and empirical evidence (ph.d. thesis). Jyvskyl Studies in Computer Science (1998) 301

[15] Henderson-Sellers, B., Debenham, J.: Towards open methodological support for agent-oriented systems development. In Far, B., Rochefort, S., Moussavi, M., eds.: Proceedings of the First International Conference on Agent-Based Technologies and Systems, University of Calgary, Canada (2003) 14–24

[16] Juan, T., Sterling, L., Winikoff, M.: Assembling agent oriented software engineering methodologies from features. In: Third International Workshop on Agent-Oriented Software Engineering, Bologna - Italy (2002)

[17] Beck, K., al.M. Beedle, van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R., Mellor, S., Schwaber, K., Sutherland, J., Thomas, D.: (Agile manifesto) http://www.agilemanifesto.org.

[18] Wells, D.: (Extreme programming - a gentle introduction) http://www. extremeprogramming.org.

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

[20] Paulk, M., Weber, C., Curtis, B.: The Capability Maturity Model for Software. Addison Wesley (1995)

[21] Ghezzi, C., Jazayeri, M., Mandrioli, D.: Fundamentals of Software Engineering. Prentice Hall International (1991)

[22] Boehm, B.: A spiral model of software development and enhancement. IEEE Computer **21** (1988) 61–72

[23] Board, I.S.: Software life cycle processes (1998)

[24] Ralyte, J., Rolland, C.: An assembly process model for method engineering. In: Proceedings of the 13th Conference on Advanced Information Systems Engineering, CAISE01, Interlaken (Switzerland) (2001)

[25] Bernon, C., Cossentino, M., Gleizes, M., Turci, P., Zambonelli, F.: A study of some multi-agent meta-models. In: Agent-Oriented Software Engineering Workshop (AOSE'04), New York (USA) (2004)