IEEE FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS (FIPA)

Design Process Documentation Template

Document title	Design Process Documentation Template		
Document number	????	**Document source**	IEEE FIPA DPDF Working Group
Document status	Preliminary	**Date of this status**	2010-06-08
Supersedes	None		
Contact	FIPA-DPDF-CHAIR@LISTSERV.IEEE.ORG		
Change history			
	Author: M. Cossentino Reviewers: A. Molesini, A. Omicini, V. Hilaire, R. Fuentes, S. DeLoach, F. Migeon, N. Bonjean, M.P. Gleizes, C. Maurel, V. Seidita, C. Bernon, V.J. Botti Navarro, J. Sudeikat, R. Wolfgang, A. Gomez, J.C. González Moreno, G. Cabri, J. Pavón., A. Garro.		

Foreword

FIPA (Foundation for Intelligent Physical Agents) is an IEEE Computer Society standards organization that promotes agent-based technology and the interoperability of its standards with other technologies.

IEEE FIPA, the standards organization for agents and multi-agent systems was officially accepted by the IEEE as its eleventh standards committee on 8 June 2005.

IEEE FIPA was originally formed as a Swiss based organization in 1996 to produce software standards specifications for heterogeneous and interacting agents and agent based systems. Since its foundations, FIPA has played a crucial role in the development of agents standards and has promoted a number of initiatives and events that contributed to the development and uptake of agent technology. Furthermore, many of the ideas originated and developed in IEEE FIPA are now coming into sharp focus in new generations of Web/Internet technology and related specifications.

In March 2005, the FIPA Board of Directors presented this opportunity to the entire FIPA membership, who unanimously voted to join the IEEE computer Society. Now, it is time to move standards for agents and agent-based systems into the wider context of software development. In short, agent technology needs to work and integrate with non-agent technologies. To this end, the IEEE Computer Society has formally accepted FIPA to become part of its family of standards committees as IEEE FIPA.

Index

1. Introduction to the specification

This work addresses process definition in a way that is similar to the one adopted by Rumbaugh[1] when addressing for the unification of object-oriented methods. Booch, Jacobson and Rumbaugh identified the common concepts among their methods, considering that basic concepts are more important than the notation used to show them. Nevertheless, notation is important to allow different people to communicate; so, after the identification of basic concepts was done, they agreed a common notation for them. This notation constitutes the Unified Modeling Language (UML).
In the same way, this specification identifies the fundamental concepts in the definition of design processes (regarding Agent-Oriented Systems) independently of the notation (text, icon, diagram, etc.) used for defining such concepts.

The design process documentation template proposed in this specification is particularly relevant also to researchers and practitioners working on Situational Method Engineering (SME)[2] approaches. SME proposes the reuse of fragments from known methods to obtain ad-hoc methods suitable for specific development situations. The method fragment (i.e. a portion of a design process) is the key-concept in SME and, although different definitions can be found for it, all of them share the idea of fragment as a self-contained component. Usually, fragments are already defined and available for use when a new design process has to be built.
The current work should be thought as previous to that moment. If fragments are to be used, they should be defined (this was partially done by FIPA Methodology TC[3]). Defining substantial fragments of a process requires the whole process to be previously described in a standard way that makes identification and definition of fragments easier. This is the main aim of this specification.
It is therefore important to provide here the means of defining the whole process from which fragments will be obtained. In this way, this specification keeps on with the work of FIPA Methodology TC addressing the initial step in definition: the proposal of a standard template in order to address process definition.

The template has some features that make it suitable for process definition.
First, it has been conceived without referring to any specific process or methodology. Moreover, the template is also neutral regarding the MAS metamodel and/or the modeling notation adopted in the process description.

---

[1] J. E. Rumbaugh. Notation notes: Principles for choosing notation. JOOP, 9(2):11– 14, 1996.

[2] S. Brinkkemper. Method engineering: engineering the information systems development methods and tools. Information and Software Technology, 37(11), 1996.
B. Henderson-Sellers. Method engineering: Theory and practice. In D. Karagiannis and editors Mayr, H. C., editors, Information Systems Technology and its Applications., pages 13–23, 2006.
I. Mirbel and J. Ralyť e. Situational method engineering: combining assembly-based and roadmap-driven approaches. Requirements Engineering, 11(1):58–78, 2006.

[3] M. Cossentino, S. Gaglio, A. Garro, and V. Seidita. Method fragments for agent design methodologies: from standardisation to research. International Journal of Agent- Oriented Software Engineering (IJAOSE), 1(1):91–121, 2007.

Secondly, the template has a simple structure resembling a tree. This allows the definition to be given in a natural and progressive way. Initially the document addresses the general description of the process, and its metamodel as well. This part is at the root of the document. Then, it details (as branches of the tree) the process phases. Next, the designer must describe thinner branches like activities or subactivites.
Such a structure can support complex processes and different situations.
Third, the template has been conceived to allow a presumably easy use by process designers with a background on software engineering. It relies only on a few initial assumptions common in the field. Moreover, the notation suggested for documenting the process is the SPEM[4] standard with few extensions. More details about the adopted notation and other useful assumptions are reported in the next subsection.

1.1. Assumptions

This document assumes several underlying ideas, which are fundamental for the understanding of the proposal. We try to make them explicit in this section.

The first assumption concerns the way of layering the design process representation. The work to be done in the process is supposed to be divided into three main levels: phase, activity and task.
Phases are composed of activities, which in turn are composed of other activities or individual, and atomic tasks. This is only a simplification used for allowing an easy catching of the correct abstraction level when documenting the process.
From a work product point of view, phases are supposed to deliver a major artefact (for instance a requirement analysis document or a detailed design document). Activities are supposed to produce finer grained artefacts (like a diagram possibly complemented by a text description of the elements reported within it). Tasks are supposed to concur to the definition of activity-level artefacts.
Such a classification is not too tight and although useful for aligning the description of very different processes, it is still open enough to accommodate all needs.

1.2. Notation

In this specification, notation is not considered fundamental, although the use of standards is important. In particular, SPEM 2.0 is suggested for modelling some process aspects.
Because of agent-oriented specific needs, some SPEM extensions are also proposed along with a few new diagrams besides the SPEM ones.

[4] O.M.G. Software & Systems Process Engineering Meta-Model Specification. Version 2.0, formal/2008-04-01. http://www.omg.org/, 2008.

In any case, this does not mean that other standards cannot be used with the template as far as the concepts implied and the underlying view of the system proposed by the work product is reflected in the notation used.
Neither specification nor suggestion is provided in this document about the modelling notation to be adopted by the documented design process. Its workflow will produce documents, diagrams and other artefacts according to the notation preferred by its designer. What is strongly advised is to think the system modelling notation as one of the possible notations to be adopted in the process and to separate its description from the description of the work product where it is adopted.

2. Design process documentation outline

The proposed documentation outline is reported in Figure 1. It is composed of three main sections (Introduction, Phases of the Process, and Work Product Dependencies). Section one and two are composed of several subsections. This structure will be detailed in the following sections according to a specific format including (for each element of the process documentation template):

- **Goal** describing the goal addressed in this part of the documentation. Example of goals include the documentation of the general philosophy that is behind a design process or the description of the involved stakeholders.
- **Structure** describing what is to be reported in this part of the process documentation. This may include diagrams as well as the textual description of specific process elements.
- **Guidelines** describing best practices suggested for a good application of the process documentation template or techniques about how to perform the prescribed work.
- **Example** addressing an existing example, possibly reported in this document.

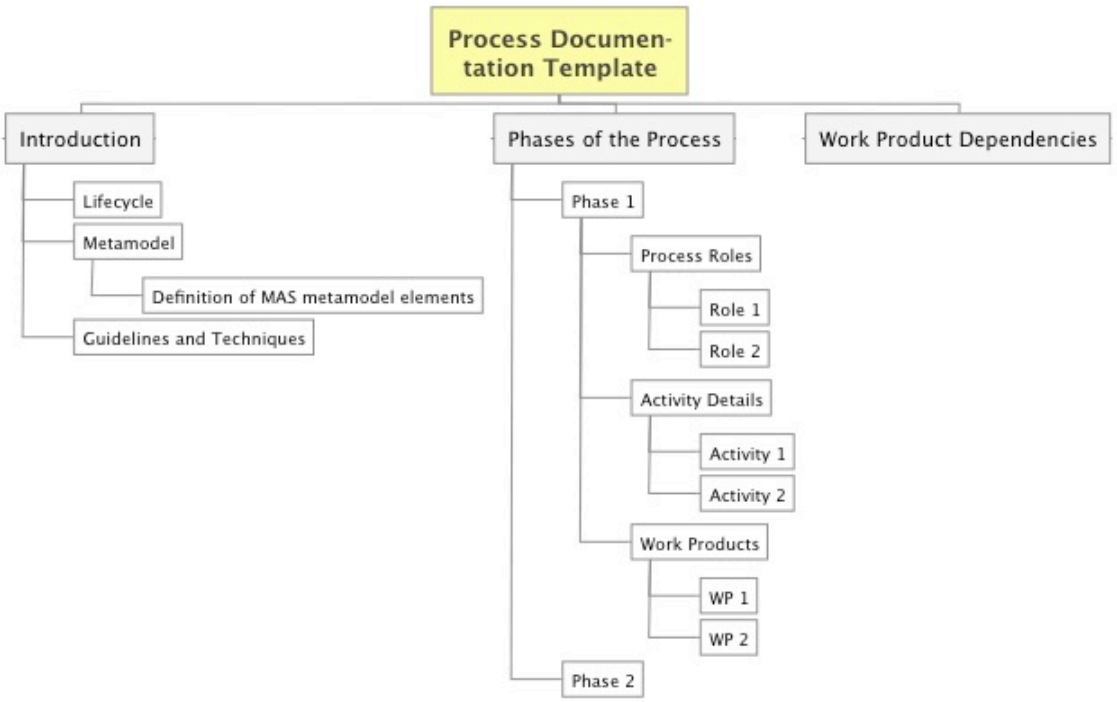

Figure 1. Design Process Documentation Outline

3. Introduction section

GOAL
This section aims at introducing the philosophy, basic ideas, scope, and limits of the process.

STRUCTURE
This section should discuss:

- concepts at the basis of the process
- a 'classic' figure of the process
- a quick description of the process (using the original process terminology if useful)
- scope of the process (kind of MAS, size, architecture, type of problems, Implementation platforms supported, ...)
- limits of the process
- reference materials and documents

EXAMPLE
See section 6.1.

3.1. Global process overview (Lifecycle)

GOAL
The aim of this section is to organize the process phases according to the selected lifecycle (or process model) in order to provide a bird-eye view of the whole process at the highest level of detail.

STRUCTURE
This section should include:

- a picture depicting the process lifecycle at the phase level and clearly showing the adopted process model (waterfall, iterative, ...)
- a description of the process phases

EXAMPLE
See section 6.1.1.

3.2. Metamodel

GOAL
The aim of this section is to provide a complete description of the MAS metamodel adopted in the process with the definition of its composing elements.

STRUCTURE
This section should include:

- An initial introduction to the basic concepts used in the metamodel. The level of granularity adopted in the metamodel should be reported as well.
- A picture of the MAS metamodel adopted in the process
- The definition of MAS metamodel elements is provided in the following subsection (see subsection 3.2.1)

GUIDELINES
Metamodeling abstraction levels are subject to the designer point of view, aim and the process philosophy.
Sometimes metamodels only address the main elements of an approach with the goal of showing the concepts at the basis of the process and their relationships.
Other times, a complete metamodel is used as a modeling notation language and this usually brings to huge (but high-fidelity) metamodels.
First kind of metamodels is sometimes addressed as 'ontological' while the others as 'linguistic'.
It is up to the author of the process documentation to establish the level of granularity s/he will adopt in this document. A recommendation is only raised in order to ensure coherence in the different parts of the document and specifically with the modeling notation described in the Work Products subsections of each phase section.

EXAMPLE
See section 6.1.2

3.2.1. Definition of MAS metamodel elements

GOAL
Providing a complete definition of the MAS metamodel adopted in the design process in terms of its composing elements.

STRUCTURE
Definitions should be provided by using a table composed of the following columns:
1) Concept: name of the MAS metamodel element
2) Definition: a complete definition of the element
3) (optional) Referred Concepts: A list of concepts used in the definition of this element and defined elsewhere in the table
4) (optional) Domain: If the metamodel is divided into domains (or layers), the name of the domain the specific concept belongs to.

EXAMPLE
See section 6.1.2.1.

3.3. Guidelines and Techniques

GOAL

The goal of this section is to provide some guidelines on how to apply the process or to clarify techniques to be used in process enactment whose validity spread throughout the whole process.

STRUCTURE

This section should provide a clear and well detailed descriptions of the guidelines and/or techniques that are at the basis of the process.
This is essentially a free-text section but diagrams, tables and other artefacts may be used when necessary.

GUIDELINES

This section should only deal with concepts related to guidelines/techniques that inspire the whole process philosophy. Concepts regarding specific phases or activities should be discussed in the corresponding process element in order to clearly localize their scope.

4. Phases of the process section

GOAL

This part of the document is composed of one section per phase. The aim is to detail the whole process by adopting a tree decomposition of it on the basis of its workflow composing elements studied at different levels of granularity (phase-activity-task).

STRUCTURE

One different section should be devoted to the discussion of each phase. The structure of each section will be detailed in the following section.

EXAMPLE

See section 6.2.

4.1. Phase 1

GOAL

Each phase is now studied from a process-oriented point of view. Workflow, work products and process roles are the centre of the discussion.

STRUCTURE

The subsection discussing each phase should:

- Introduce the phase workflow by using a SPEM activity diagram. It reports activities that compose the phase, and it includes a quick description of work

products and roles.

- Introduce a SPEM diagram reporting the structural decomposition of the activity in terms of the involved elements: tasks, roles and work products

This initial introduction is followed by three subsections:

1) Process Roles
2) Activity Details
3) Work Products

(details about their structure in the following subsections)

GUIDELINES

The process is supposed to be described at three different levels of granularity regarding the work to be done: phase, activity, task.

Phase level work is supposed to deliver a composite work product (or a set of WPs) whose size is like that of a 'System Analysis document' or 'Detailed Design document'

Activities are supposed to deliver a major work product like a diagram or the code of a software component or the textual list of requirements

Tasks are supposed to concur to the development of a major work product.

Tasks are regarded as atomic units. This assumption has to be adopted according to the goal of providing a complete and clear description of the process. In many processes tasks could be further decomposed but this would bring no significant advantage to process documentation.

EXAMPLE

See section 6.2.1

4.1.1. Process roles

GOAL

The aim of this section is listing the roles involved in the work of this phase and clarifying their level of involvement in the job to be done.

STRUCTURE

This subsection should describe the responsibilities of each process role in the activities composing this phase. Roles can be responsible for activities or assist in them. The different levels of responsibility (responsible/assistant) should be clearly stated.

GUIDELINE

Adopting a common taxonomy of process roles could encourage process sharing and the reuse of their portions (fragments). A list of roles has been proposed in the paper by V. Seidita et al. (2008)[5].

[5] V. Seidita, M. Cossentino, S. Gaglio. Using and Extending the SPEM Specifications to Represent Agent Oriented Methodologies. In Agent-Oriented Software Engineering 2008. Lecture Notes in Computer Science, volume 5386-0086, pp. 46-59. Springer-Verlag GmbH. 2009.

EXAMPLE
See section 6.2.1.1.

4.1.1.1. Role 1

(description of Role 1 as discussed above)

EXAMPLE
See section 6.2.1.1.1

4.1.2. Activity details

GOAL
The aim of this section is to detail the work to be done at each activity (decomposing it with further elements of a lower level of abstractions if needed)

STRUCTURE
One subsection for each activity describing activities/tasks composing the selected activity by using a SPEM activity diagram including the involved roles (as swimlanes). Further details about each activity can be provided in additional sections.

EXAMPLE
See section 6.2.1.2.

4.1.2.1. Activity 1

GOAL
Describe the work to be done within this activity

STRUCTURE
Details of tasks and sub-activities are specified with a table that includes the following columns:
- Activity: name of the activity studied in this subsection.
- Tasks/Sub-Activity: sub-activity or task described in this row of the table.
- Task/Sub-activity Description.
- Roles involved.

Optionally, the control flow within a Task can be illustrated by a stereotyped UML Activity Diagram. These diagrams explain the execution of complicated Tasks by denoting the possible sequences of Steps, which are identified by the <<steps>> stereotype. Details on this modelling of Tasks can be found in the current SPEM specification.

When documenting a Task in this way, the diagrams are appended and each diagram is discussed in a separated paragraph that explains the illustrated steps and their relations.

EXAMPLE
See section 6.2.1.2.1

4.1.3. Work products

GOAL
The aim of this section is twofold. The first part aims at detailing the information content of each work product by representing which MAS model elements are reported in it and which operations are performed on them. The second part focuses on the modeling notation adopted by the process in the specific work product.

STRUCTURE
Work products produced in this phase are described by using a work product structure diagram.
This diagram (an extension to SPEM specifications proposed by V. Seidita et al.[6]) is a structural (i.e. class) diagram reporting the main work product delivered by the phase and then the composing (or other not related) diagrams around that.
The structure of each composing diagram is then expressed in terms of the MAS metamodel elements defined/related/refined/quoted in it.

A table is used to describe the scope of each work product.
The table has 3 columns:
1) Name: name of the work product
2) Description: a description of the content
3) Work Product Kind: classification of the work product according to the already cited paper (categories: Free/Structured Text, Behavioural, Structural, and Composite)

The structure of the subsection devoted to detail the notation of each work product will be presented starting from subsection. 4.1.3.1.

EXAMPLE
See section 6.2.1.3.

[6] V. Seidita, M. Cossentino, S. Gaglio. Using and Extending the SPEM Specifications to Represent Agent Oriented Methodologies. In Agent-Oriented Software Engineering 2008. Lecture Notes in Computer Science, volume 5386-0086. Springer-Verlag GmbH. 2009.

4.1.3.1. Work product kinds

GOAL
This section aims at describing the different work products according to the already presented classification

STRUCTURE
The description is done by using a table with three columns:

1. Name: name of the work product
2. Description: a short description of the artefact
3. Work product kind: can be one of the above mentioned kinds (i.e. Free Text, Structured Text, Behavioural, Structural, and Composite)

EXAMPLE
See section 6.2.1.3.1.

4.1.3.2. WP 1

GOAL
The aim of this subsection is allowing an easy adoption of the modelling approach proposed by the original process to the reader of this document or conversely, providing the creator of a new modelling notation with a clear idea of what is required in order to properly support the modelling demands of this document.

STRUCTURE:
A subsection is now reported for each different artefact in order to describe the notation and the template of the document as it is normally suggested by the original process.

The structure of this subsection includes:
1) A description of the document with a specific mention of the notation adopted.
2) Examples of the document parts. These examples also include tables, diagrams, and outlines of specific portions of text used for describing the design.

EXAMPLE
See section 6.2.1.3.2.

5. Work product dependencies section

GOAL:
The goal of this document is providing a representation of the dependencies among the work products and therefore (indirectly) among the activities that produce them.

This can prove very important to project manager who have to reschedule project activities according to new need occurring at design time.

STRUCTURE
This is a classic diagram (also specified by SPEM). All the work products produced by the process are reported in the diagram and a dashed arrowed line is used to relate two of them if one is an input document to the other. The direction of the arrow points from the input document to the consumer one.

It is to be noted that according to the importance that is paid to the MAS metamodel in the Agent-Oriented Software Engineering (AOSE) field, the real input of each portion of process is a subset of model elements (instances of the MAS metamodel) that constitute the portion of design reported in the input documents.

EXAMPLE
See section 6.3.

6. Appendix: example of documentation

This section reports an example of application of the Process Documentation Template to the PASSI process.

6.1. Introduction

PASSI (Process for Agent Societies Specification and Implementation) is a step-by-step requirement-to-code methodology for designing and developing multi-agent societies. The methodology integrates design models and concepts from both Object-Oriented software engineering and artificial intelligence approaches.

PASSI has been conceived in order to design FIPA-compliant agent-based systems, initially for robotics and information systems applications.

Systems designed by using the PASSI process are usually composed of peer-agents (although social structures can be defined). According to FIPA specifications agents are supposed to be mobile, and they can interact by using semantic communications referring to an ontology and an interaction protocol.

PASSI is suitable for the production of medium-large MAS (up to a hundred agent-kinds each one instantiated in an unlimited number of agents in the running platform).

The adoption of patterns and the support of specific CASE tools (PTK) allows a quick and affordable production of code for the JADE platform. This encourages the use of this process even in time/cost-constrained projects or where high quality standards have to be met.

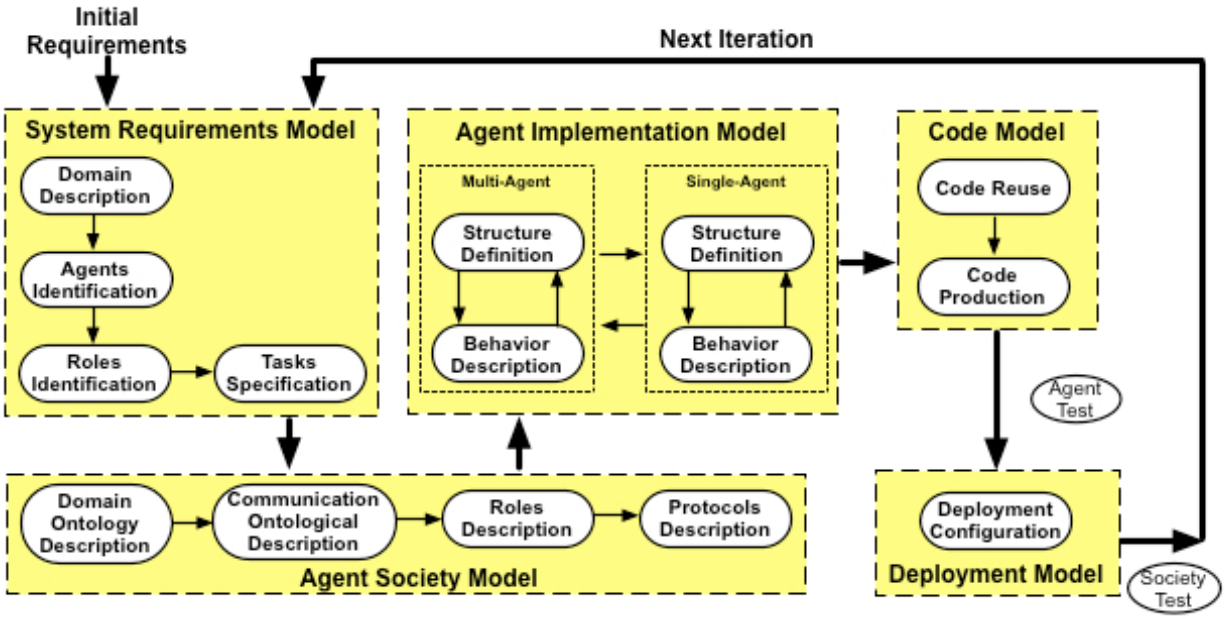

Figure 2. The PASSI design process

The design process is composed of five models (see Figure 2): the System Requirements Model is a model of the system requirements; the Agent Society Model is a model of the agents involved in the solution in terms of their roles, social interactions, dependencies, and ontology; the Agent Implementation Model is a model of the solution architecture in terms of classes and methods (at two different levels of abstraction: multi and single-agent); the Code Model is a model of the solution at the code level and the Deployment Model is a model of the distribution of the parts of the system (i.e. agents) across hardware processing units, and their movements across the different available platforms.

In the following the PASSI process will be described by initially considering its whole process and then its five components, each of them representing a phase, a portion of work for which a specific outcome and milestones can be identified and represented in the following diagram.

Useful references about the PASSI process are the following:

- M. Cossentino. From Requirements to Code with the PASSI Methodology. In Agent-Oriented Methodologies, B. Henderson-Sellers and P. Giorgini (Editors). Idea Group Inc., Hershey, PA, USA. 2005.
- M. Cossentino, S. Gaglio, L. Sabatucci, and V. Seidita. The PASSI and Agile PASSI MAS Meta-models Compared with a Unifying Proposal. Lecture Notes in Computer Science, vol. 3690. Springer-Verlag GmbH. 2005. pp. 183-192.
- M. Cossentino and L. Sabatucci. Agent System Implementation in Agent-Based Manufacturing and Control Systems: New Agile Manufacturing Solutions for Achieving Peak Performance. CRC Press, April 2004.
- M. Cossentino, L. Sabatucci, and A. Chella. Patterns reuse in the PASSI methodology. In Engineering Societies in the Agents World IV, 4th International Workshop, ESAW 2003, Revised Selected and Invited Papers, volume 3071 of Lecture Notes in Artificial Intelligence. Springer-Verlag, 2004. pp. 294-310
- M. Cossentino, L. Sabatucci, A. Chella - A Possible Approach to the Development of Robotic Multi-Agent Systems - IEEE/WIC Conf. on Intelligent Agent Technology (IAT'03). October, 13-17, 2003. Halifax (Canada)
- Chella, M. Cossentino, and L. Sabatucci. Designing JADE systems with the support of case tools and patterns. Exp Journal, 3(3):86-95, Sept 2003.

Useful references about PASSI extensions are the following:

- M. Cossentino, N. Gaud, V. Hilaire, S. Galland, A. Koukam. ASPECS: an Agent-oriented Software Process for Engineering Complex Systems. International Journal of Autonomous Agents and Multi-Agent Systems (IJAAMAS). 20(2). 2010.
- M. Cossentino, G. Fortino, A. Garro, S. Mascillaro and W. Russo. PASSIM: a simulation-based process for the development of MASs. International Journal on Agent Oriented Software Engineering (IJAOSE), 2(2). 2009.
- V. Seidita, M. Cossentino, S. Gaglio. Adapting PASSI to Support a Goal Oriented Approach: a Situational Method Engineering Experiment. Proc. of the Fifth

European workshop on Multi-Agent Systems (EUMAS'07). 13-14 December, 2007. Hammameth (Tunisia).

- A. Chella, M. Cossentino, L. Sabatucci, and V. Seidita. Agile PASSI: An Agile Process for Designing Agents. International Journal of Computer Systems Science & Engineering. Special issue on "Software Engineering for Multi-Agent Systems", 21(2). March 2006.

6.1.1. The PASSI Process lifecycle

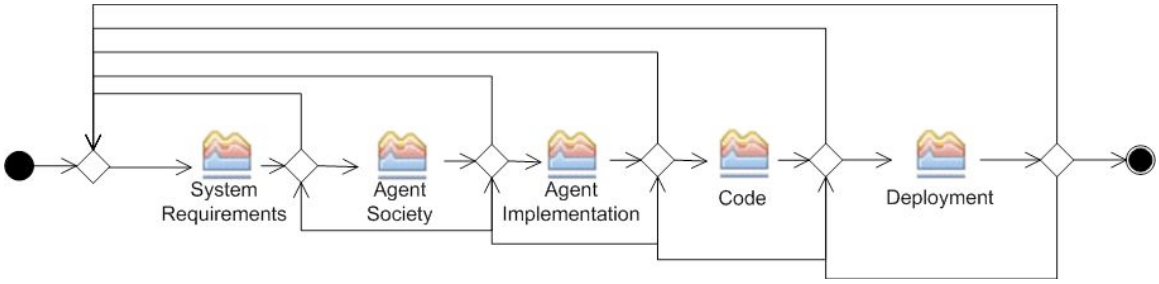

Figure 3. The PASSI process phases (and iterations)

PASSI includes five phases arranged in an iterative/incremental process model (see Figure 3):

- System Requirements: It covers all the phases related with requirements elicitation, requirements analysis and agents/roles identification
- Agent Society: All the aspects of the agent society are addressed: ontology, communications, roles description, and interaction protocols
- Agent Implementation: A view on the system's architecture in terms of classes and methods to describe the structure and the behaviour of single agents.
- Code: A library of class and activity diagrams with associated reusable code and source code for the target system.
- Deployment: How the agents are deployed and which constraints are defined/identified for their migration and mobility.

Each phase produces a document that is usually composed aggregating UML models and work products produced during the related activities. Each phase is composed of one or more sub-phases each one responsible for designing or refining one or more artefacts that are part of the corresponding model. For instance, the System Requirements model includes an agent identification diagram that is a kind of UML use case diagrams but also some text documents like a glossary and the system use scenarios.
The details of each phase will be discussed in the following section.

6.1.2. The PASSI MAS metamodel

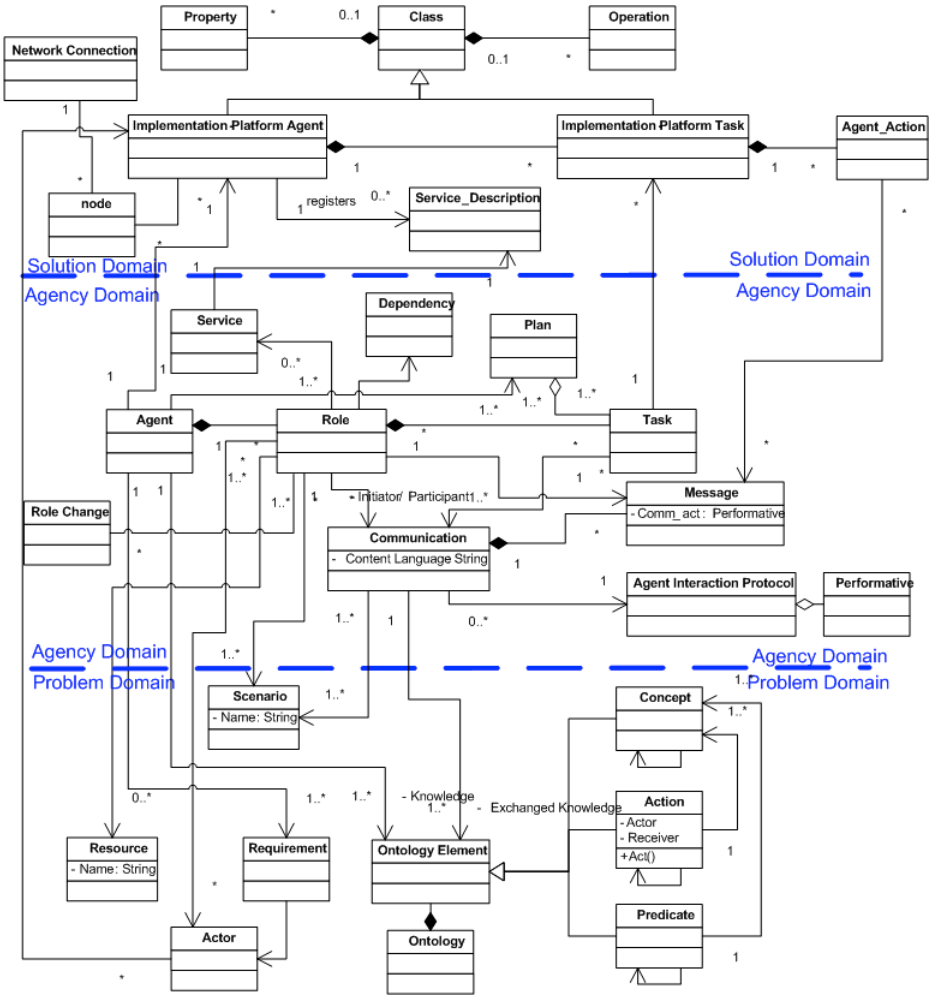

Figure 4. The PASSI MAS metamodel

The description of the PASSI MAS meta-model (MMM) (Figure 4) addresses three logical areas: (i) the problem domain, (ii) the agency domain, and (ii) the solution domain.

In the problem domain the designer includes components describing the requirements the system is going to accomplish: these are directly related to the requirements analysis phase of the PASSI process.

Then the designer introduces the agency domain components which are used to define an agent solution for the problem.
Finally, in the PASSI MMM solution domain, agency-level components are mapped to the adopted FIPA-compliant implementation platform elements. Here it is supposed the platform supports at least the concepts of agent and task. This represents the code-level part of the solution and the last refinement step.

Going into the details of the model (see Figure 4), the Problem Domain deals with the user's problem in terms of scenarios, requirements, ontology and resources. Scenarios describe a sequence of interactions among actors and the system to be built.
The ontological description of the domain is composed of concepts (categories of the domain), actions (performed in the domain and effecting the status of concepts) and predicates (asserting something about a portion of the domain, i.e. the status of concepts).
Resources are the last element of the problem domain. They can be accessed/shared/manipulated by agents. A resource could be a repository of data (like a relational database), an image/video file or also a good to be sold/bought.
The Agency Domain contains the components of the agent-based solution. None of these components is directly implemented; they are converted to the correspondent object-oriented entity that constitutes the real code-level implementation.
The key concept of this domain is the agent. An agent is responsible for realizing some functionalities descending from one or more functional requirements. It has also to respect some non functional requirement constraints (like for instance performance prescriptions). The agent is a situated entity that lives in an environment from which it receives perceptions (the related knowledge is structured according to the designed domain ontology). Sometimes an agent has also access to available resources and it is capable of actions in order to pursue its own objectives or to offer services to the community.
The functionality of an agent is organized through roles. Each agent during its life plays some roles. A role is a peculiarity of the social behaviour of an agent. When playing a role, an agent may provide a service to other agents executing tasks and processing messages.
A task specifies the computation that generates the effects of a specific agent behavioural feature. It is used with the significance of an atomic part for defining the overall agent's behaviour. This means that an agent's behaviour can be composed by assembling its tasks and the list of actions that are executed within each task cannot be influenced by the behaviour planning. Tasks are structural internal components of an agent and they contribute to define the agent's abilities. These tasks cannot be directly accessed by other agents unless the agent offers them as a set of services, as agents are autonomous entities.
A communication is an interaction among two agents and it is composed of one or more messages seen as speech acts. The information exchanged during a communication is composed of concepts, predicates or actions defined in the ontology. The flow of messages and the semantics of each message are ruled by an agent interaction protocol (AIP).
The last Agency Domain element is the service. It describes a set of coherent functionalities exported by the agent for the community.
The Implementation Domain describes the structure of the code solution in the chosen FIPA-compliant implementation platform. It is essentially composed of three elements: (i) the FIPA-Platform Agent that is the base class catching the implementation of the Agent entity represented in the Agency domain; (ii) the FIPA-Platform Task that is the

implementation of the agent's Task, (iii) the ServiceDescription component that is the implementation-level description (for instance an OWL-S[7] file) of each service specified in the Agent Domain.

6.1.2.1. Definition of MAS metamodel elements

Concept	Definition	Domain
Requirement	A requirement represents a feature that the system-to-be must exhibit. It can be a functional requirement such as a service or a non-functional requirement, such as a constraint on the system (or a specific part of it) performance.	Problem
Actor	An entity (human or system) interacting with the agents system.	Problem
Service	A service is a single, coherent block of activity in which an agent will engage. A set of services can be associated with each agent role.	Agency
Agent	We consider two different aspects of the agent: during the initial steps of the design, it is seen (this is the Agency Domain Agent) as an autonomous entity capable of pursuing an objective through its autonomous decisions, actions and social relationships. This helps in preparing a solution that is later implemented referring to the agent as a significant software unit (this is the Solution Domain FIPA-Platform Agent). More in details, an Agent is an entity that: - is capable of action in an environment; - can communicate directly with other agents typically using an Agent Communication Language; - is driven by a set of functionalities it has to accomplish; - possesses resources of its own; - is capable of perceiving its environment; - has only a partial representation of this environment in form of an instantiation of the domain ontology (knowledge); - can offer services; - can play several different (and sometimes concurrent or mutually exclusive) roles.	Problem
Role	A portion of the social behaviour of an agent that is characterized by a goal (accomplishing some specific functionality) and/or provides a service.	Problem
Task	A task specifies the computation that generates the effects of the behavioural feature. Its granularity addresses the significance of a non decomposable group of atomic actions that cannot be directly addressed without referring to their belonging task.	Problem

[7] http://www.w3.org/Submission/OWL-S/

Communication	An interaction among two agents, referring an Agent Interaction Protocol and a piece of the domain ontology (knowledge exchanged during the interaction). Usually it is composed of several messages, each one associated with one Performative.	Agency
Ontology, concept, action, predicate	An ontology is an explicit specification of the structure of a certain domain. Therefore it provides a vocabulary for representing and communicating knowledge about some topic and a set of relationships and properties that hold for the entities denoted by that vocabulary.	Agency
Implementation Platform Agent	The software implementation of the Agent in the selected platform	Solution
Implementation Platform Task	The software implementation of the Task in the selected platform	Solution

Note that optional column *Referred Concepts* described in 3.2.1 has not been used in the above table.

6.2. Phases of the PASSI Process

6.2.1. The System Requirements Phase

The process flow at the level of activities is reported in Figure 5. The process flow inside each activity will be detailed in the following subsections (after the description of process roles).

The System Requirements phase involves two different process roles, eight work products (four UML models and four text documents) and four guidance documents (one for each UML model) as described in the following Figure 6.

The phase is composed of four activities (i.e. Domain Requirements Description, Agents Identification, Roles Identification and Task Specification), each of them composed of one or more tasks (for instance Identify Use Cases and Refine Use Cases).

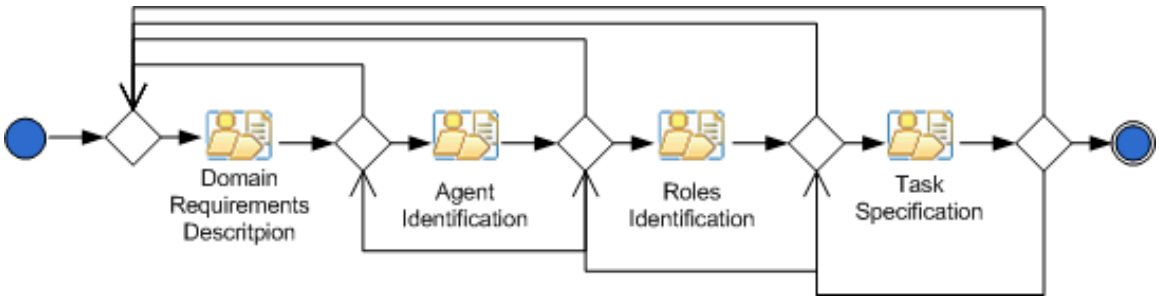

Figure 5. The System Requirements Phase flow of activities

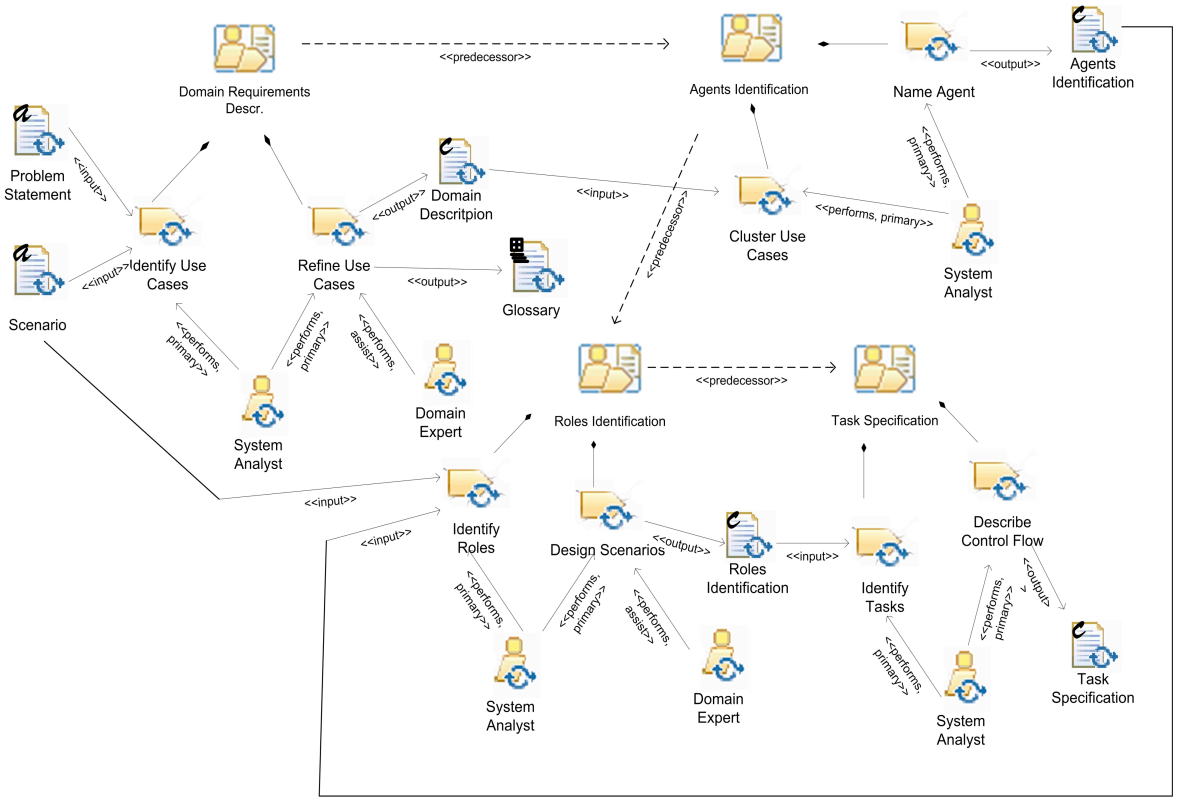

Figure 6. The System Requirements phase described in terms of activities and work products

6.2.1.1. Process roles

Two roles are involved in the System Requirements discipline: the System analyst and the Domain expert. They are described in the following subsections.

6.2.1.1.1. System analyst

S/he is responsible of:

1. Use cases identification during the Domain Requirements Description (DRD) activity. Use cases are used to represent system requirements.
2. Use cases refinement during the DRD activity. Use cases are refined with the help of a Domain Expert.
3. Use cases clustering during the Agent Identification (AID) activity. The System Analyst analyzes the use case diagrams resulting from the previous phase and attempts to cluster them in a set of packages.
4. Naming agents during the AID activity. After grouping the use cases in a convenient set of packages, the last activity of this phase consists in designing these packages with the names that will distinguish the different agents throughout all the project.
5. Roles identification during the Role Identification (RId) activity. The System Analyst studies (textual) scenarios and system requirements (as defined in the previous phase) and identifies the roles played by agents.

6. Designing scenarios during the RID activity. Each scenario in designed in form of sequence diagrams thus depicting the details of agents interactions
7. Tasks identification during the Task Specification (TSP) activity. It consists in the identification of the behavioural capabilities that each agent needs to perform the specified roles and the fulfil the requirements that are under its responsibility.
8. Description of the control flow during the TSP activity. It consists in introducing the communication relationships among tasks of different agents and the control flow among tasks of the same agent.

6.2.1.2. Activity Details

6.2.1.2.1. Domain Requirements Description

The flow of tasks inside this activity is reported in Figure 7 and the tasks are detailed in the following table.

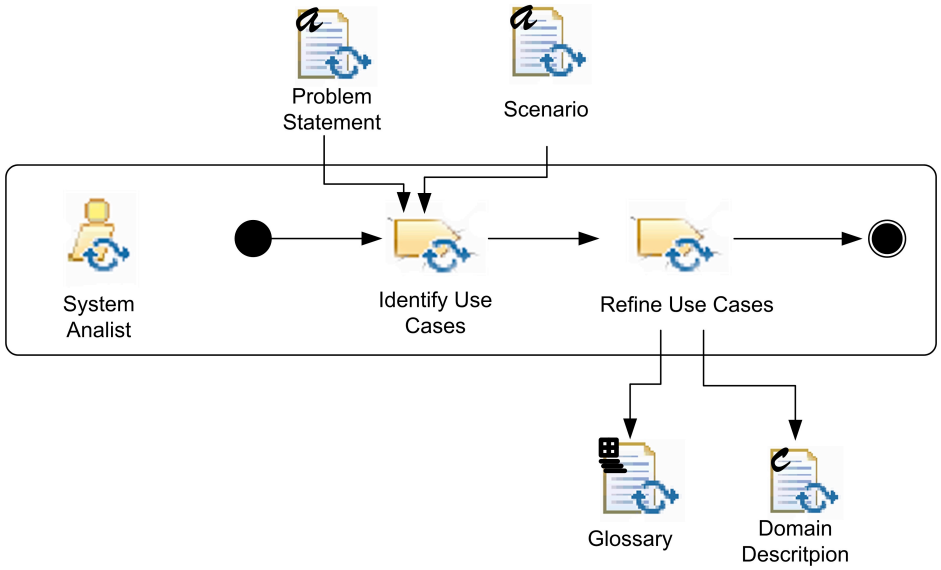

Figure 7. The flow of tasks of the Domain Requirements Description (DRD) activity

Activity	Task	Task Description	Roles involved
Domain Requirements Description	Identify Use Cases	Use cases are used to represent system requirements	System Analyst (perform)
Domain Requirements Description	Refine Use Cases	Use cases are refined with the help of a Domain Expert	System Analyst (perform) Domain Expert (assist)

6.2.1.3. Work Products

The System Requirements Model generates four composed work products (text documents including diagrams in this case). Their relationships with the MAS meta-model elements are described in the following Figure 8.

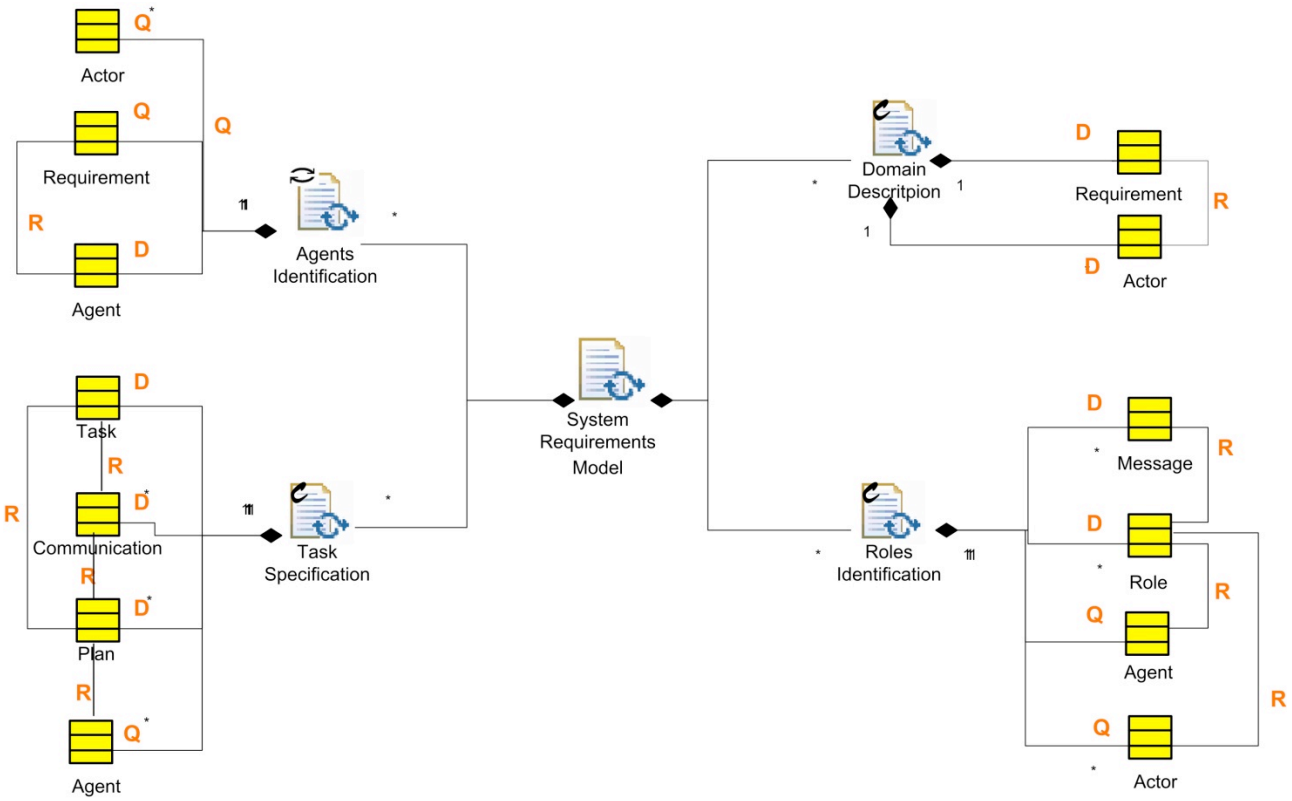

Figure 8. The System Requirements Model documents structure

This diagram represents the System Requirement model in terms of Work Products. Each of these reports one or more elements from the PASSI MAS meta-model; each MAS meta-model element is represented using an UML class icon (yellow filled) and, in the documents, such elements can be Defined, reFined, Quoted, Related or Relationship Quoted.

6.2.1.3.1. Work Product Kinds

Name	Description	Work Product Kind
Problem Statement	A description of the problem to be solved with the system. It is complemented by the Scenario document	Free Text

Scenarios	Textual description of the scenarios in which the system to be developed is involved	Free Text
Domain Description	A text document composed by the Domain Description diagram, a documentation of use cases reported in it and the non-functional requirements of the system	Composite (Structured + Behavioural)
Agent Identification	A document composed of: 1) a use case diagram representing agents and the functionalities assigned to them; 2) a structured text description of the agents	Composite (Structured + Behavioural)
Roles Identification	A document composed of several sequence diagrams (one for each scenario) and the roles description text	Composite (Structured + Behavioural)
Task Specification	A document composed of several Task specification diagrams (one for each agent) and a structured text description of each task	Composite (Structured + Behavioural)
Glossary	A glossary of terms	Structured Text
Code	Code of the software solution	Structured Text

6.2.1.3.2. Agent Identification

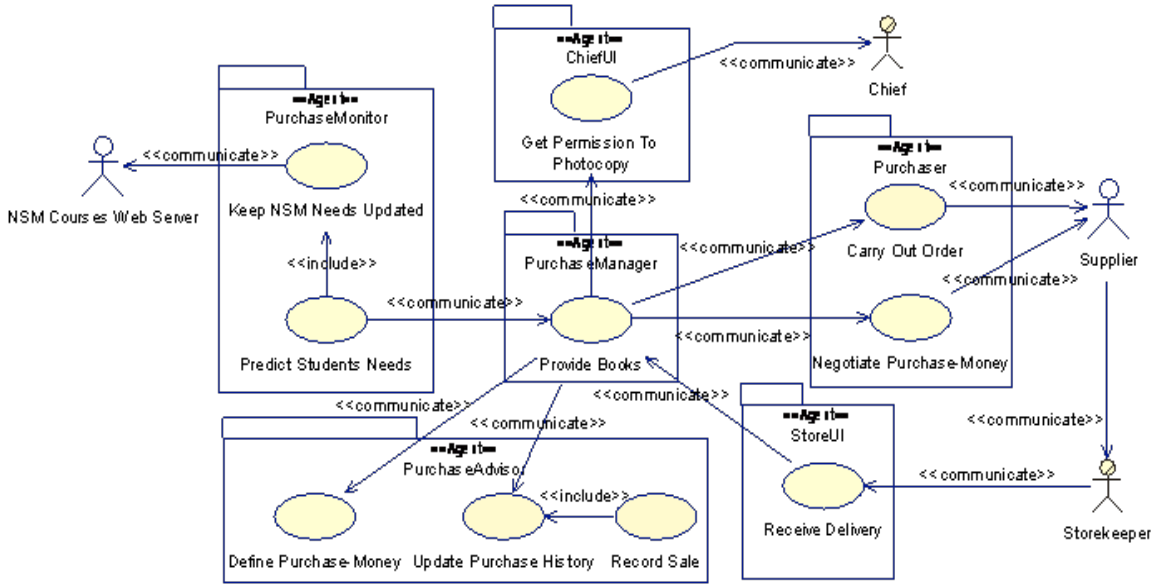

Figure 9. PASSI Agent Identification diagram

Starting from a use case diagram, packages are used to group functionalities that will be assigned to an agent (whose name is the name of the package).
Stereotypes of relationships between use cases of different packages (agents) are converted to 'communicate' since different agents can interact only in this way. Direction of the relationships goes from the initiator to the participant in the communication.

The diagram is completed by the following information:

Agent	Description (Functional Req.)	Special Requirements
		(security, other non funct. Req., pseudo-req, mobility, …)

6.3. Work product dependencies

This diagram describes the dependencies among the different work products. For instance, the Communication Ontology diagram depends on the Domain Ontology diagram since during the specification of the communications parameters it is necessary to know the ontology elements (concepts, actions, predicates) defined in the previous phase.

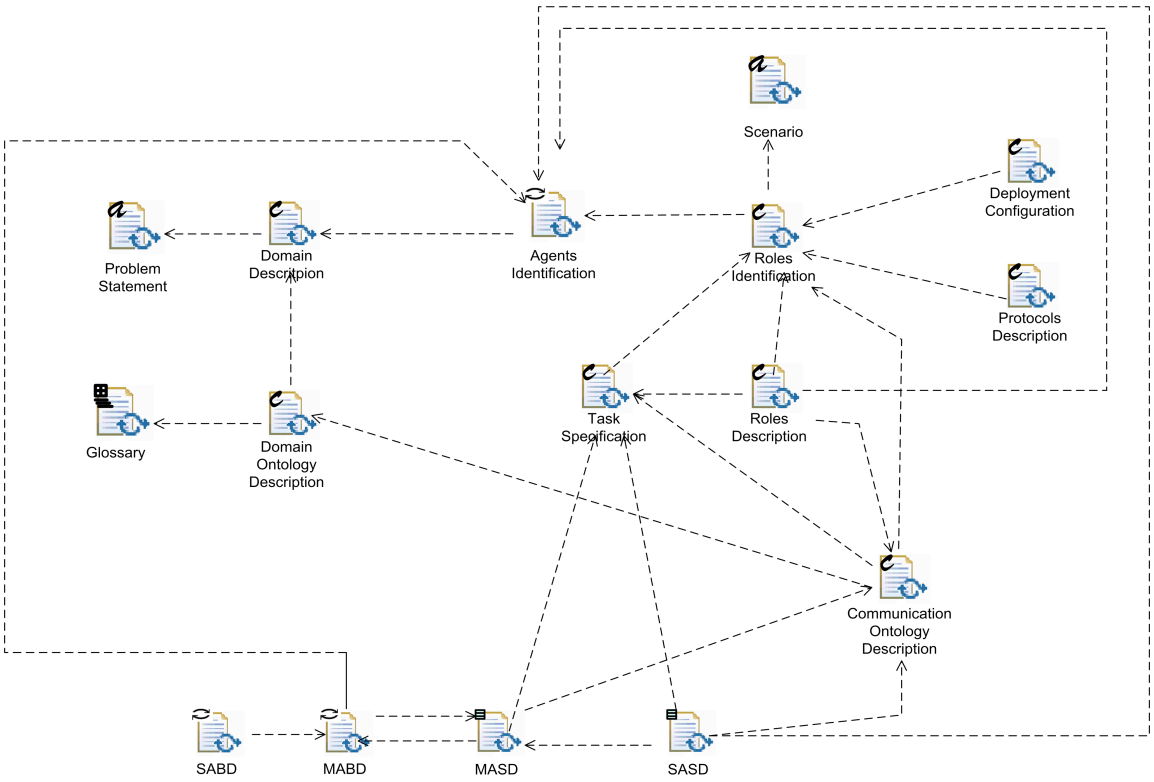

Figure 10. Work product dependency diagram