IEEE FIPA Design Process Documentation and Fragmentation Working Group

Fragment Definition and Documentation Template

Atomic/Composed/Phase Process Fragment

(Initial) Author(s): M. Cossentino, V. Seidita, A. Molesini, V. Hilaire

Contributors: (TBD)

Last saved on: 05/05/12 15:05

Index

1. Introduction to the specification

It is currently admitted in both mainstream software engineering and agent- oriented software engineering that there is no one-size-fit-all methodology or process.
One solution to overcome this problem is proposed by the Situational Method Engineering SME) paradigm [4][5][6].
It consists in providing means for constructing ad-hoc software engineering processes following an approach based on the reuse of portions of existing design processes, the so called method fragments, stored in a repository called method base.
One problem raised by this type of approaches is to describe or document fragments and to choose among existing fragments in order to build a new process.

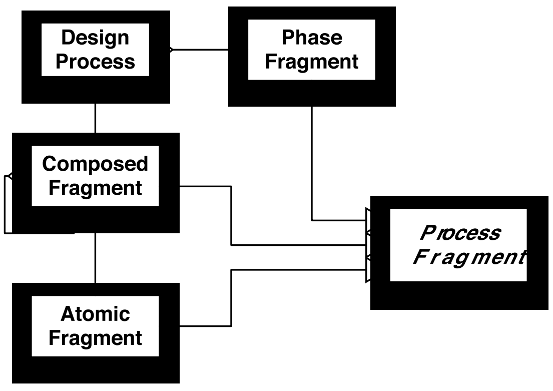
1.1. Assumptions

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 documenting design process, the reference standard is 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 fragment. 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. As in the following documentation template SPEM 2.0 is used as process modelling language, sometimes the use of term task overlap with that of activity.

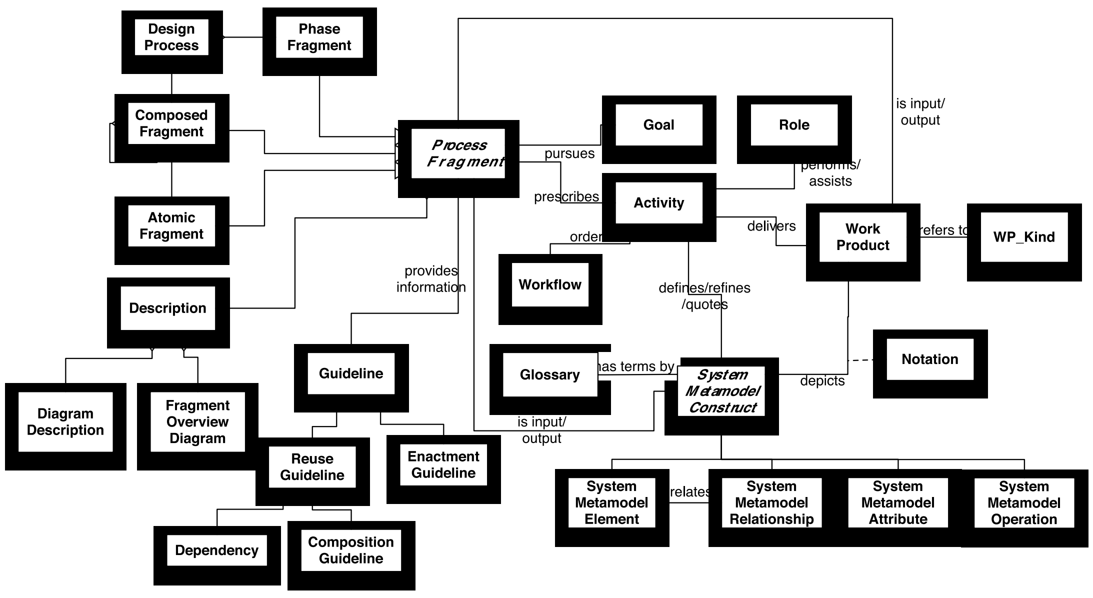
Such a classification is not too tight and although useful for aligning the description of very different processes, it is still open enough to accommodate all needs.

The second assumption concerns the definition of *Process Fragment*, it is a portion of design process adequately created and structured for being reused during the composition of new design processes both in the field of agent oriented software engineering and in other ones (model driven engineering-based approaches are preferred fields of application for the proposed definition). The process fragment is, generally (this is the most common case), extracted from an existing design process and it is stored in a repository.
The previous classification regarding phase, activity and task let the method engineer identify and extract, from design process, fragments belonging to different levels of granularity; three different levels have been identified: phase, composed and atomic, each of them is related to the quantity of work to be done and to the complexity of the produced outcome.
Figure NUMBER shows the relationships among design process and fragments of different level of granularity.



A **phase** (process) fragment delivers a set of work products belonging to the same design abstraction level of the design flow. Such a work product may belong to any of the cited work product types. An examples of phase-level work product may be a system analysis document; it is composed of several work products (diagrams, text documents, . . .) all belonging to the same design abstraction level (system analysis).
A **composed** (process) fragment delivers a work product (or a set of instances of the same work product). Such a work product may belong to any of the cited work product types.
An **atomic** (process) fragment delivers a portion of a work product and/or a set of system model constructs (in terms of their instantiation or refinement). A portion of a work product is here intended never to be a whole work product; in other words, atomic fragments never deliver entire work products [7].

Figure NUMBER represents all the elements composing a Process Fragment. It contains all the elements useful for representing and documenting the fragment under the process, product and reuse point of view; the proposed fragment documentation template that will be presented in the following section slavishly follows the proposed representation, its elements and their definitions.



The root element, the *Process Fragment*, has been generally extracted from an existing design process, therefore an important information to be stored in the repository is the

Design Process the fragment refers to. This serves for the designer to set the application context and the particular features the fragment would exhibit.
The process fragment prescribes some activities to do, each of them is a portion of work that has to be performed by one or more stakeholders (*Roles*).
Activity delivers *Work Products*, where the results of design activities are drawn by using a specific *Notation* and each work product is developed under the responsibility of one role. The notation to be used greatly influences the flow of work to be done for producing a work product and for this reason a fragment has to be supplied with a set of *Guidelines*. As regards the process and product perspective of the fragment the *Enactment Guidelines* provides all the elements, description and so on, for applying the workflow prescribed in the fragment.
It is not mandatory to follow a specific notation; the same kind of diagram (for instance a structural one) may be expressed by using different notations without significant differences in the resulting expressiveness. Moreover, different kinds *WP_Kind*) of work products can be delivered.
We identified two main work product kinds: graphical and textual, the former when an activity results in a diagram, the second when designers produce textual documents.
Finally a work product can be of composite kind if it is a composition of the previous said kinds, for instance a document with a diagram and the text explaining it (more details can be found in aoseSeiditaCG08).
As well as in the design process definition, one of the most important elements in the fragment definition is the (Multi-Agent) System Metamodel (Multi-Agent SMM); each fragment is based on a system metamodel that is obviously a part of the metamodel of the design process it comes from. As regard the system metamodel issues (see for details [4]) the following definitions have to considered:
A system metamodel is the set of constructs, and their definitions, needed for creating system models. Such constructs are: elements, relationships, attributes and operations. A system metamodel element (SMME hence after) is an entity of the metamodel that is instantiable in an entity of the system model. A System Metamodel Relationship (SMMR hence after) is used to represent the existence of a relationship between two (or more) instances of SMMEs. System metamodel attributes and operations are used for respectively representing a particular feature and the behavioral characteristics of an element.
Beside, for better representing and documenting the process fragment, we consider three different kinds of metamodel:

- the **Complete System Metamodel** includes all the system metamodel constructs that are managed by the designers using a specific design process. This also includes all the constructs that are accepted as external inputs in the process.
- the **Definable System Metamodel** includes all the system metamodel constructs that are instantiated during the design process enactment. This is a subset of the complete system metamodel.
- the **Workproduct System Metamodel** includes all the complete system metamodel constructs that are reported in the design process work products.

The main aim of process fragment is to instantiate one (or more) system metamodel construct(s) (SMMC) and in so doing it may be requested to define relationships among elements or to quote other elements and/or relationships; besides the result of defining an element or a relationship might be the refinement of existing elements or relationships.
This fact led to the definition of the kinds of action to be done on a system metamodel construct (see the following section for details). Finally SMMC has a definition to be listed in a glossary; the definition is mainly useful during selection when the method designer wants

know which kind of metamodel construct better fits with the metamodel construct s/he is dealing with.
Until now we explored the process and product part of the fragment through a set of elements that has to be necessarily present in the fragment documentation, now let us quickly focus on the elements that principally deal with the reuse aspect of the fragment: *Goal*, and *Dependency* guidelines.
The fragment goal is the objective the process part of the fragment wants to pursue and it is to be used during fragment selection from the repository. For this reason it is related to the new design process requirements, in other words, a goal describes the contribution a fragment may give to the accomplishment of some design process requirements.

The dependency guideline aims at describing specific constraints, if they exist, for the fragment to be composed with other ones, for instance, there can be fragments dealing with system metamodel elements that are very specific to particular application domains, in this case it should be possible that such fragments can be composed with fragments coming from the same classes of design processes.
It is important noting that the way the work has to be performed inside one fragment may slightly change depending on the notation of the work product produced; if the result has to be a graphical work product the activity and the related guidelines are different if we want to use two different notations. Since the fragment aims at designing a specific system metamodel construct, we can consider the fragment itself independent from the specific notation. The same result can be obtained by producing different work products in different notations.

In the following table we give the detailed definition of all the elements composing a process fragment:

Term	**Definition**
Design Process	It is the design process from which the fragment has been extracted.
Phase	A specification of the fragment position in the design workflow. Usually referring to a taxonomy (i.e. Requirements elicitation, Analysis, Design, etc.)
Goal	The process-oriented objective of the fragment.
Activity	A portion of work assignable to a performer (role). An activity may be atomic (sometimes addressed as Action) or composed by other activities.
Work Product	The resulting product of the work done in the fragment; it can be realized in different ways (diagram, text,..) also depending on the specific adopted notation.
WP_Kind	Represents the specific kind one work product can be; it strictly depends on the means the adopted notation provides. One work product can be: Structured or Free text, Structural, Behavioural or Composite
Notation	Each deliverable can be drawn by using a specific notation. Concepts dealt by the fragment have to find a mapping in the notation. Notation usually includes a metamodel and a set of pictorial prescriptions used to represent the instantiation of metamodel elements.
Role	The stakeholder performing the work in the process and responsible of producing a work product (or a part of it). Usually referring to a taxonomy (i.e. System Analyst, Test Designer, etc.)
System Metamodel Construct	(abstract class) The concept the fragment deals with, for instance a fragment aiming at defining the system requirements has to define and to identify the concept of “requirements”. Each metamodel construct has to be defined

	during, at least, one portion of process work and has to appear in at least one work product.
System Metamodel Element	It is an entity of the metamodel that is instantiable into an entity of the system model. Examples of System Metamodel Elements (SMME) are: classes, use cases,….
System Metamodel Relationship	It is the construct used for representing the existence of a relationship between two (or more) instances of SMMEs. For instance, the aggregation relationship among two instances of a SMME class is an instance of the SMMR association.
System Metamodel Attribute	It is a particular kind of elements used for adding properties to SMMEs. *An SMMA is a structural feature and it relates an instance of the class to a value or collection of values of the type of the attribute.*\cite{UML2.2}. The attribute's type is a SMME.
System Metamodel Operation	It is a behavioral feature of a classifier that specifies the name, type, parameters, and constraints for invoking an associated behaviour
Glossary	A list of definitions for the system metamodel constructs.
Description	It is the textual and pictural description of the fragment; it provides a bird-eye on the whole process the fragments comes from and the fragment overview in terms of tasks to be performed, roles and work product kind to be delivered.
Composition Guideline	A set of guidelines for assembling/composing the fragments with others. This may include notational specifications, and constraints (also addressing issues like platform to be used for system implementation and application area).
Dependency	The description of specific dependencies of this fragment from other ones; it is useful for composition.
Enactment Guideline	The description of how to perform the prescribed activity. This may include best practices and specific techniques for achieving the expected results.

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 fragment aspects.
Because of agent-oriented specific needs, some SPEM extensions are also proposed along with a few new diagrams besides the SPEM ones.
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. Fragment documentation outline

The proposed documentation outline is composed of seven main sections: Description, System Metamodel, Stakeholders, Workflow, Deliverables, Guidelines, References. 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 document. Example of goals include the documentation of the general philosophy that is behind a fragment or the description of the involved stakeholders.
- **Structure** describing what is to be reported in this part of the process document. This may include diagrams as well as the textual description of specific fragment elements.
- **Guidelines** describing best practices suggested for a good application of the fragment documentation template or techniques about how to perform the prescribed work.
- **Example** addressing an existing example, possibly reported in this document.

3. Fragment Documentation Template

3.1. Fragment Description section

3.1.1. Fragment Goal

GOAL
The aim of this section is to provide the reader with a quick understanding of the goal pursued by the fragment, possibly relating the description to common-sense in software engineering (i.e.: the aim of this fragment is collecting requirements in a text form)
STRUCTURE: Free text
GUIDELINES: --
EXAMPLE: See Section 4.1.1

3.1.2. Fragment Granularity

GOAL
The aim of this section is indicating the level of granularity of the fragment.
STRUCTURE
Free text such as the following: Granularity of this fragment: (Phase/Composed/Atomic)
GUIDELINES
It is possible to decide among three different levels of granularity: *phase, composed and atomic*, for a complete description see section 0
EXAMPLE: See Section 4.1.2

3.1.3. Composing fragments

GOAL
The aim of this section is to provide the list of fragments this fragment is composed of. For instance if this fragment is a composed fragment it is reasonable to think that it is composed of several atomic fragments.

STRUCTURE
This section should include text and a table as the one shown in the following:

This fragment is composed of the following fragments:

Composing Fragment name	Granularity	Process of Origin

GUIDELINES
For each composing fragment it useful to indicate its level of granularity and the Design Process it comes from.
EXAMPLE: See Section 4.1.2.1

3.1.4. Fragment Origin

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 4.1.3

3.1.4.1. The Process 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 4.1.3.1

3.1.5. Fragment Overview

GOAL
This section aims at illustrating the structural composition of this fragment in terms of tasks

to be performed, roles and work product kind to be delivered.

STRUCTURE
This section should provide a SPEM 2.0 activity diagram.

EXAMPLE: See Section 4.1.4

3.2. System metamodel section

GOAL
This section provides the user with a detailed description of the portion of the complete system metamodel managed by the fragment.
STRUCTURE
A structural diagram (usually a UML class diagram) depicting the system metamodel elements and their relationships

GUIDELINES
In drawing this class diagram we should consider all the elements, relationship, operation and attribute that are managed in the fragment workflow, both the ones that serve only as input and that effectively reported in the document. For a detailed description about how to identify system metamodel constructs and the rules for drawing the workproduct content diagram (to be designed in section 3.5.2) see [4]
EXAMPLE: See Section 3.2

3.2.1. Definition of System metamodel elements

GOAL: Provide a precise definition of all the system metamodel elements of this fragment
STRUCTURE: a table with two columns (name and definition) or a structured text (vocabulary style)
EXAMPLE: See Section 4.2.1

3.2.2. Definition of System metamodel relationships

GOAL: Provide a precise definition of all the system metamodel relationships of this fragment
STRUCTURE a table with two columns (name and definition) or a structured text (vocabulary style)
EXAMPLE: See Section 4.2.2

3.2.3. Definition of System metamodel attributes

GOAL: Provide a precise definition of all the system metamodel attributes of this fragment
STRUCTURE a table with two columns (name and definition) or a structured text (vocabulary style)
EXAMPLE: See Section 4.2.3

3.2.4. Definition of System metamodel operations

GOAL: Provide a precise definition of all the system metamodel operations of this fragment
STRUCTURE a table with two columns (name and definition) or a structured text (vocabulary style)

EXAMPLE: See Section 4.2.4

3.2.5. Fragment Input/Output in Terms of System Metamodel Constructs

GOAL
Provide a precise view on all the input of the fragment in terms of system metamodel elements, relationships, attributes and operations and all the outputs detailing which are to be designed, refined or quoted.

STRUCTURE
This section should provide a table as the following, where only the construct name has to be inserted.

	Input	**To Be Designed**	**To Be Refined**	**To Be Quoted**
SMME				
SMMR				
SMMA				
SMMO				

EXAMPLE: See Section 4.2.5

3.3. Stakeholders section

GOAL
The aim of this section is listing the roles involved in the fragment workflow.
STRUCTURE
The subsection should describe the responsibilities of each role. Role can be the principal responsible of a task or he can assist. The different level of responsibility (performs, assists) should be clearly stated.

GUIDELINES
Adopting a common taxonomy of process roles could encourage process sharing and the reuse of their portion (fragments). A list of roles has been proposed in Seidita et al. [6]
EXAMPLE: See Section 4.3.

3.3.1. Role 1

(*description of Role 1 as discussed above)*

EXAMPLE
See section 4.3.1.

3.4. Workflow section

3.4.1. Workflow description

GOAL
This section aims at detailing the work to be performed in each

STRUCTURE
This subsection provides a SPEM 2.0 activity diagram where for each stakeholder the flow of tasks and the related workproducts are reported.
GUIDELINES
EXAMPLE: See Section 3.4.1

The process that is to be performed in order to obtain the result is represented in the following as a SPEM 2.0 diagram.

3.4.2. Activity description

GOAL
The aim of this subsection is to provide detailed information on each activity of the process fragment workflow
STRUCTURE
The structure of this subsection should be in form of a table like the following one.
The fragment encompasses the following *work breakdown elements*:

Name	Kind	Description	Roles involved

GUIDELINES

EXAMPLE: See Section

3.4.3. Fragment's activities input/output in terms of system metamodel constructs

GOA
The aim of this subsection is to highlight, for each task and or activity of the process fragment,
STRUCTURE
GUIDELINES
EXAMPLE: See Section …

The above described *work breakdown elements* have the following input/output in terms of system metamodel components.
In the Input column, system metamodel components utilization is completed by the name of the input document reporting them in the original design process.

Input				
Activity/Task Name	**SMME**	**SMMR**	**SMMA**	**SMMO**

Output				
Activity/Task Name	**SMME**	**SMMR**	**SMMA**	**SMMO**

3.4.4. Fragment's Input/Output in terms of Work Product

GOAL
STRUCTURE
GUIDELINES
EXAMPLE: See Section …

Input, output work products to be designed in the fragment are detailed in the following tables.

Input	Output

3.5. Deliverables section

3.5.1. Document name

GOAL
STRUCTURE
GUIDELINES
EXAMPLE: See Section …

3.5.1.1. Deliverable notation

3.5.1.1.1. Example

3.5.2. Deliverable content in terms of system metamodel constructs

GOAL
STRUCTURE
GUIDELINES
EXAMPLE: See Section …

3.6. Guidelines section

3.6.1. Enactment Guidelines

GOAL
STRUCTURE
GUIDELINES
EXAMPLE: See Section …

--

3.6.2. Reuse Guidelines

GOAL
STRUCTURE
GUIDELINES
EXAMPLE: See Section …

3.6.2.1. Composition

--

3.6.2.2. Dependency Relationship with other fragments

3.7. References section

GOAL
STRUCTURE
GUIDELINES
EXAMPLE: See Section …

4. Appendix: Example of Process Fragment Documentation

This section reports an example of application of the Fragment Documentation template to Domain Requirements Description (DRD) fragment from PASSI design process.

4.1. Fragment Description

4.1.1. Fragment Goal

Representing system functional and non-functional requirements

4.1.2. Fragment Granularity

Granularity of this fragment: composed

4.1.2.1. Composing fragments

This fragment is composed of the following fragments:

Composing Fragment name	Granularity	Process of Origin

4.1.3. Fragment Origin

The presented fragment has been extracted from *PASSI* (Process for Agent Societies Specification and Implementation) design process.
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.

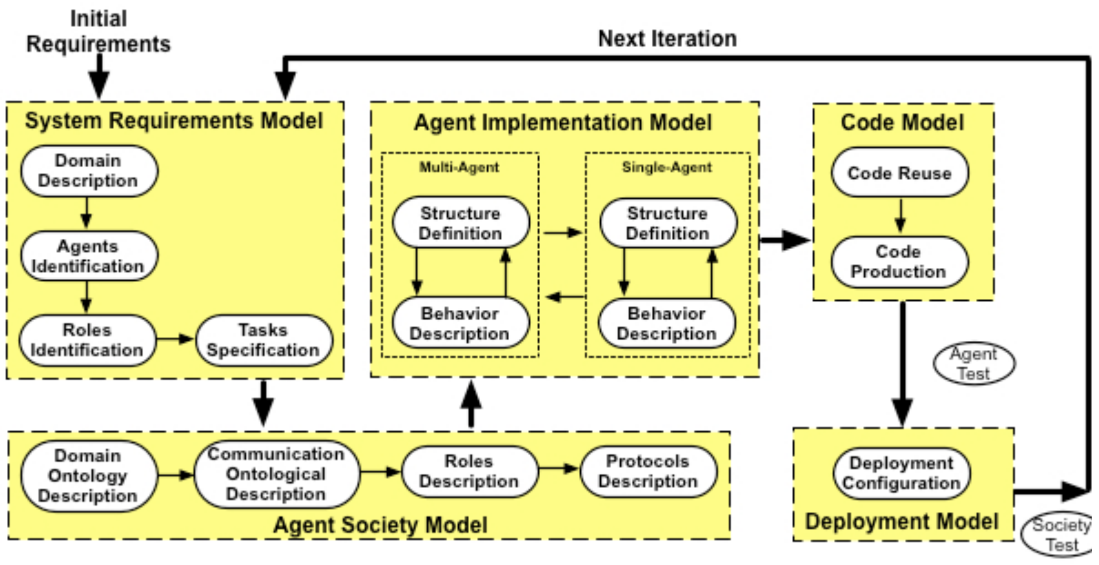


Figure 1. The PASSI design process

The design process is composed of five models (see Figure 1): 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.

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.

4.1.3.1. The PASSI Process lifecycle

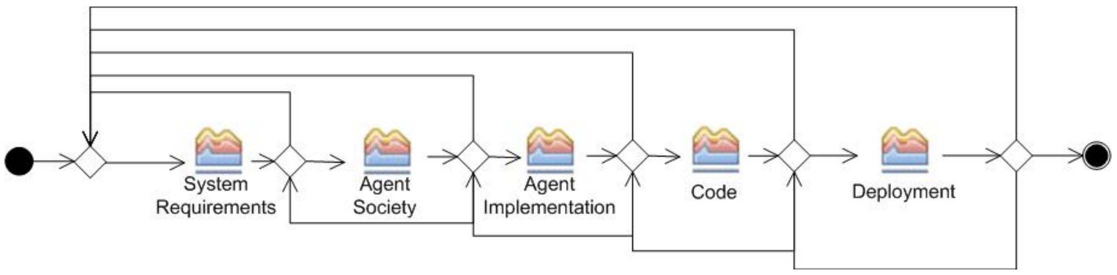


Figure 2. The PASSI process phases

PASSI includes five phases (see Figure 2) arranged in an iterative/incremental process model:

- System Requirements: It covers all the phases related to Req. Elicitation, analysis and agents/roles identification
- Agent Society: All the aspects of the agent society are faced: ontology, communications, roles description, Interaction protocols
- Agent Implementation: A view on the system’s architecture in terms of classes and methods to describe the structure and the behavior of single agent.
- 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.

4.1.4. Fragment Overview

Consider the PASSI process (Figure 1) and the “System Requirements” phase with its outcome “System Requirements Model”. Now, consider the “Domain Requirements Description” (red colored in Figure 3) activity and the consequent outcome (the “Communication Ontological Description” composite document).

This activity aims to model the social interactions and dependencies among the agents involved in the solution and to face the following agent society aspects are faced: communication and role description. The activity and its main outcome has been considered for being extracted from PASSI and for becoming a process fragment.

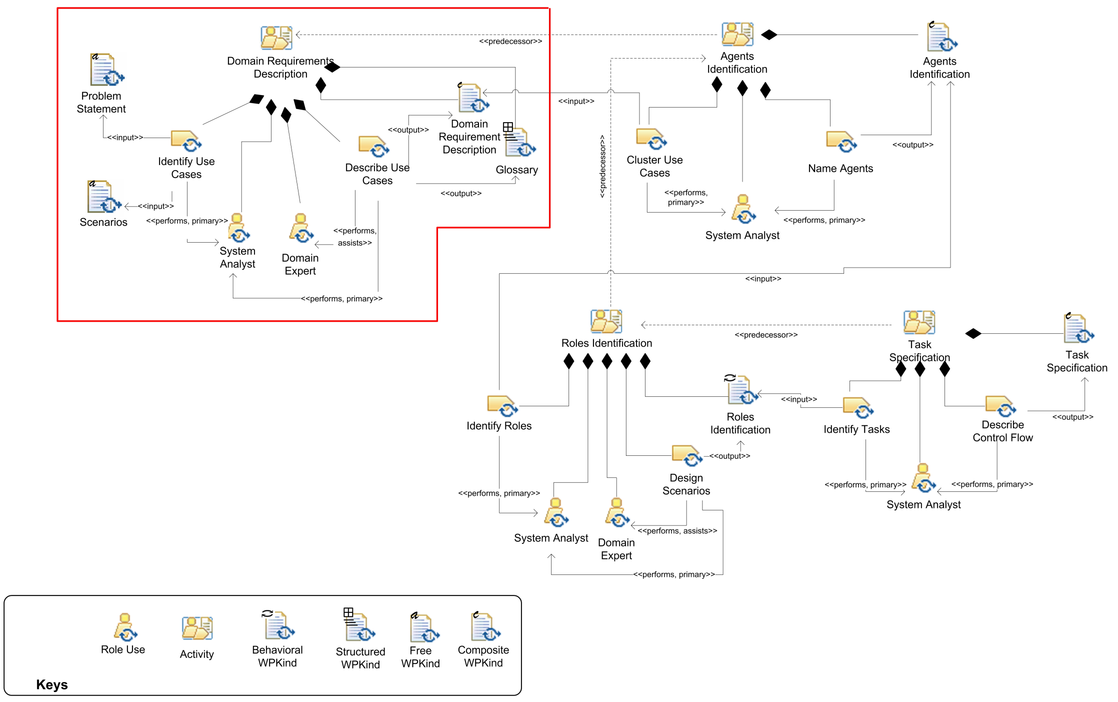


Figure 3. The System Requirements Phase (structural view)

4.2. System metamodel

The portion of metamodel of this fragment is:

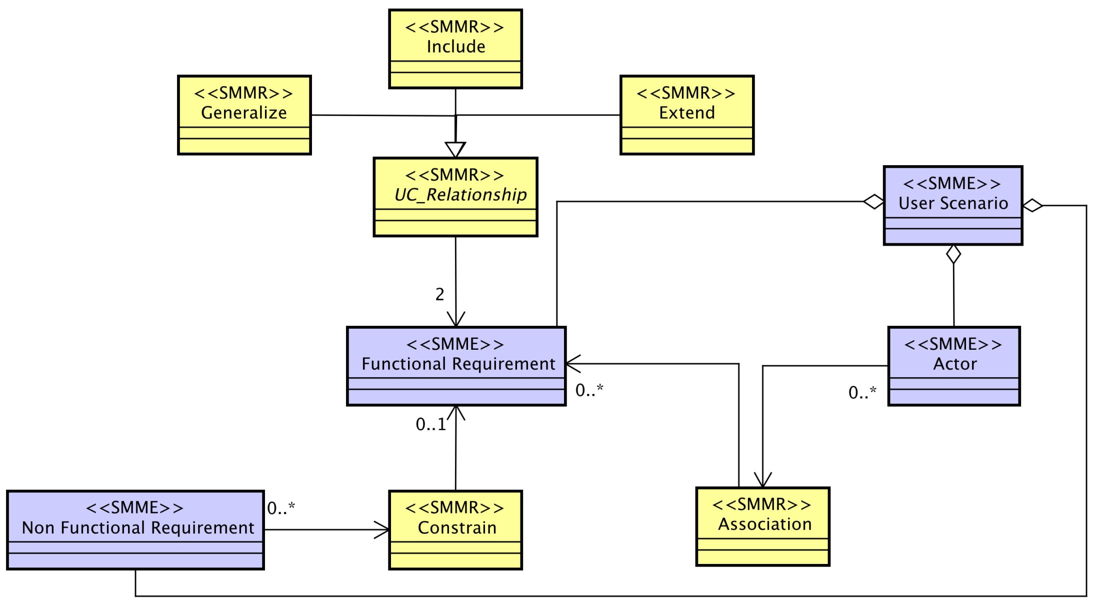


Figure 4. The fragment system metamodel

This fragment refers to the MAS metamodel adopted in PASSI and contributes to define and describe the elements reported in Figure 4.

4.2.1. Definition of System metamodel elements

This fragment underpins the following model elements:

Functional requirement - Functional requirements describe the functions that the software is to execute. (from IEEE SEBOK 2004)
Non-Functional requirement - Non functional requirements constrain the solution and are sometimes known as constraints or quality requirements. (from IEEE SEBOK 2004)
Actor - An external entity (human or system) interacting with the multi-agent system.
User Scenario: -"A narrative description of what people do and experience as they try to make use of computer systems and applications" [M. Carrol, Scenario-based Design, Wiley, 1995]

4.2.2. Definition of System metamodel relationships

Generalize – (see UML definition)
Include – (see UML definition)
Extend – (see UML definition)
Association – (see UML definition)
Constrained_by – It relates a Functional Requirement to a Non Functional Requirement. It means that a functionality of the system has to be realised under some non functional constraints (see FURPS+ to have examples of non functional requirements).

4.2.3. Definition of System metamodel attributes

None

4.2.4. Definition of System metamodel operations

None

4.2.5. Fragment Input/Output in Terms of System Metamodel Constructs

Input, output system metamodel constructs to be designed in the fragment are detailed in the following table.

	Input	To Be Designed	To Be Refined	To Be Quoted
SMME	User Scenario	• Actor • Functional Requirement • Non Functional Requirement		
SMMR		• Functional Requirement-Functional Requirement (Generalize, Include, Extend) • Functional Requirement-Actor (Association)		
SMMA				
SMMO				

4.3. Stakeholder

Roles involved in this fragment are:

- System Analyst
- Domain Expert

Their responsibilities are described in the following subsections.

4.3.1. System Analyst

He is responsible for:

1. Use cases identification
2. Use cases refinement. Use cases are refined with the help of a Domain Expert.

4.3.2. Domain Expert

1. He supports the system analyst during the description of the domain requirements.

4.4. Workflow

4.4.1. Workflow description

The process that is to be performed in order to obtain the result is represented in the following as a SPEM2.0 diagram.

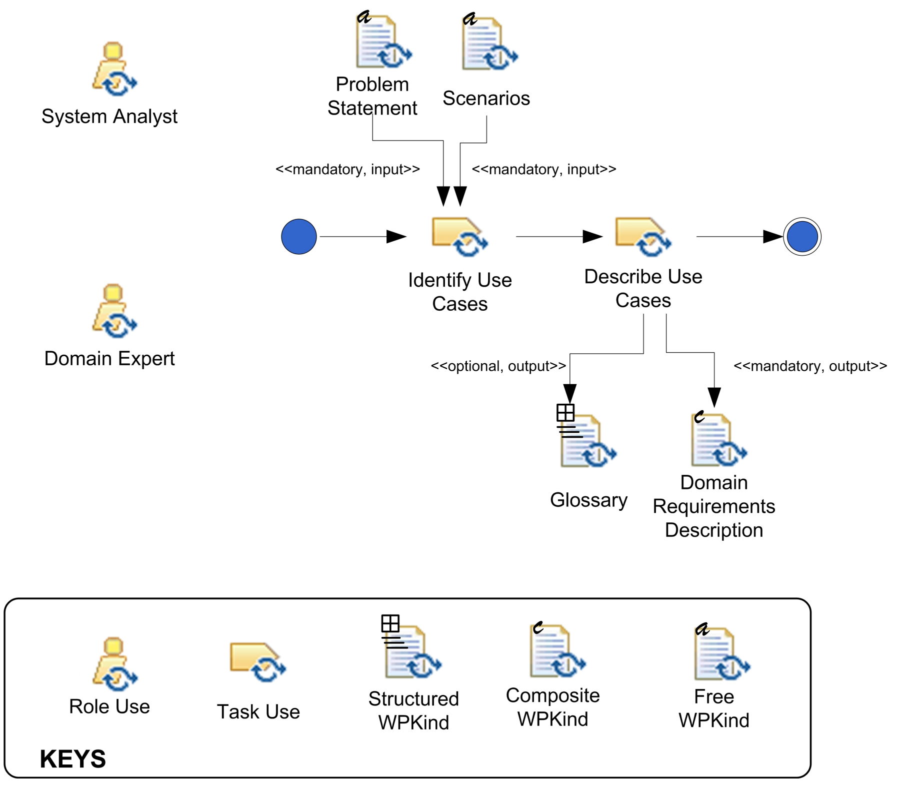


Figure 5. The flow of tasks of this process fragment

4.4.2. Activity description

The fragment encompasses the following *work breakdown elements*:

Name	Kind	Description	Roles involved
Identify Requirements	Task	It consists in identifying use cases in order to represent the system requirements.	System Analyst (performs)
Describe Requirements	Task	Use cases are described with the help of a Domain Expert	System Analyst (performs), Domain Expert (assists)

4.4.3. Fragment's activities input/output in terms of system metamodel constructs

The above described *work breakdown elements* have the following input/output in terms of system metamodel constructs.

Input				
Activity/Task Name	**SMME**	**SMMR**	**SMMA**	**SMMO**
Identify Requirements	• Scenario			

Output				
Activity/Task Name	**SMME**	**SMMR**	**SMMA**	**SMMO**
Describe Requirements	• Actor • Functional Requirement • Non Functional Requirement.	• Functional Requirement-Functional Requirement (Generalize, Include, Extend) • Functional Requirement-Actor (Association) • Functional Requirement-Non Functional Requirement (Constrained By)		

4.4.4. Fragment's Input/Output in terms of Work Products

Input, output work products to be designed in the fragment are detailed in the following table.

Input	**Output**
▪ Problem Statement ▪ Scenarios	▪ System Requirements document ▪ Glossary

4.5. Deliverable

4.5.1. Domain Requirements Description Document

This fragment produces a composite document composed of use case diagrams and portions of (structured) text containing the complete documentation of the use cases in terms of: name, participating actors, entry condition, flow of events, exit condition, exceptions and special requirements.
It also reports the non functional requirements identified for the system and associated to each use case.

4.5.1.1. Domain Requirements Description Diagram: example of notation

Common UML use case diagram(s) are used to represent the system requirements.

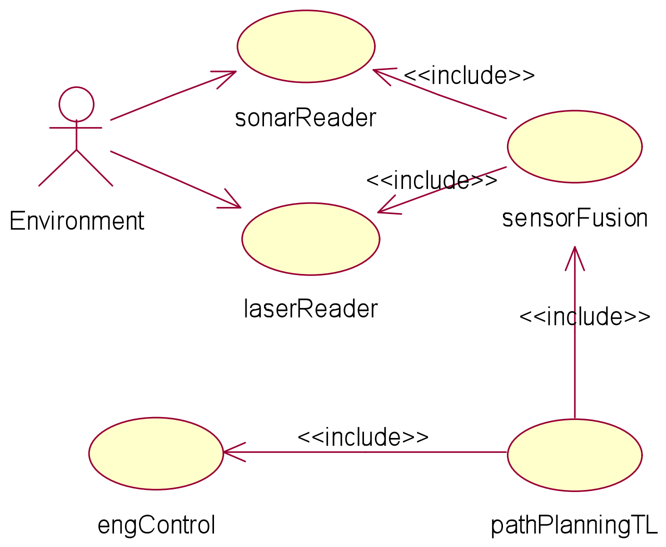
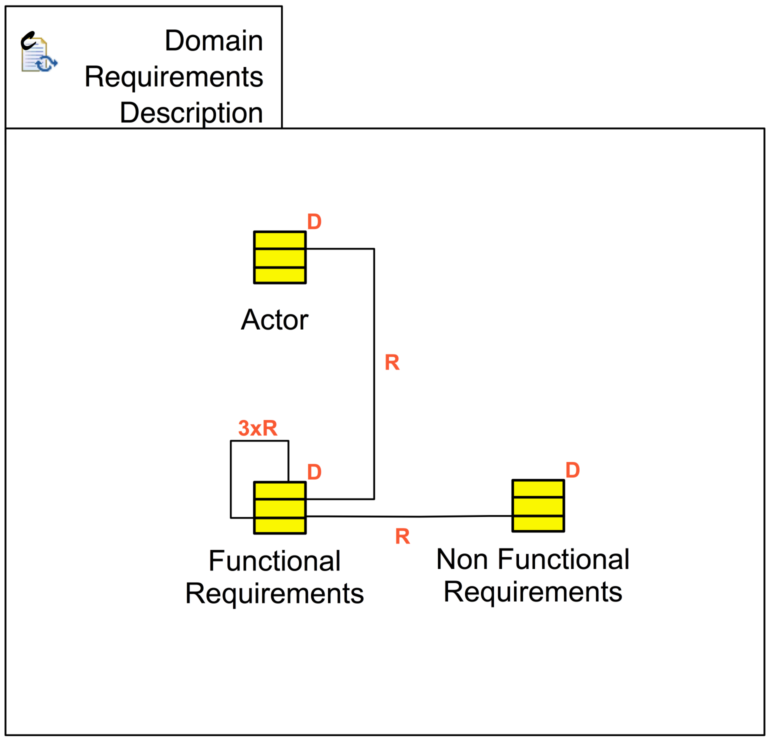


Figure 6. An example of Domain Requirements Description diagram

4.5.2. Deliverable content in terms of system metamodel constructs

The following figure describes the structure of this fragment work products in relationship with the system metamodel constructs:



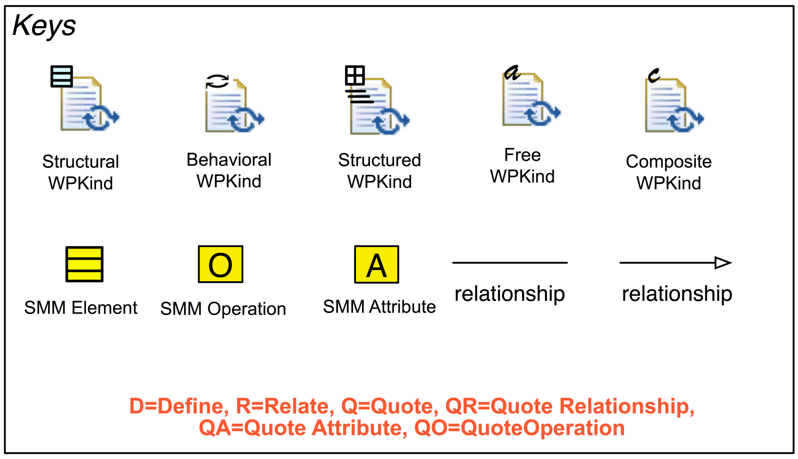


Figura 7. Structure of the fragment work-product in terms of system metamodel constructs

4.6. Guidelines

4.6.1. Enactment Guidelines

(extracted from OpenUP website, Guideline: Identify and Outline Actors and Use Cases, online at: http://epf.eclipse.org/wikis/openup/practice.tech.use_case_driven_dev.base/guidances/guidelines/identify_and_outline_actors_and_ucs_BB5516A9.html)

Identifying actors

Find the external entities with which the system under development must interact. Candidates include groups of users who will require help from the system to perform their tasks and run the system's primary or secondary functions, as well as external hardware, software, and other systems.

Define each candidate actor by naming it and writing a brief description. Includes the actor's area of responsibility and the goals that the actor will attempt to accomplish when using the system. Eliminate actor candidates who do not have any goals.

These questions are useful in identifying actors:

Who will supply, use, or remove information from the system?
Who will use the system?
Who is interested in a certain feature or service provided by the system?
Who will support and maintain the system?
What are the system's external resources?
What other systems will need to interact with the system under development?

Identifying use cases

The best way to find use cases is to consider what each actor requires of the system. For each actor, human or not, ask:

What are the goals that the actor will attempt to accomplish with the system?
What are the primary tasks that the actor wants the system to perform?
Will the actor create, store, change, remove, or read data in the system?
Will the actor need to inform the system about sudden external changes?
Does the actor need to be informed about certain occurrences, such as unavailability of a network resource, in the system?
Will the actor perform a system startup or shutdown?
Understanding how the target organization works and how this information system might be incorporated into existing operations gives an idea of system's surroundings. That information can reveal other use case candidates.

Give a unique name and brief description that clearly describes the goals for each use case. If the candidate use case does not have goals, ask yourself why it exists, and then either identify a goal or eliminate the use case.

Outlining Use Cases

Without going into details, write a first draft of the flow of events of the use cases identified as being of high priority. Initially, write a simple step-by-step description of the basic flow of the use case. The step-by-step description is a simple ordered list of interactions between the actor and the system.

As you create this step-by-step description of the basic flow of events, you can discover alternative and exceptional flows.

(The following text is largely taken from the OpenUP website page reported below but it has been also modified in some parts: http://epf.eclipse.org/wikis/openup/practice.tech.use_case_driven_dev.base/guidances/guidelines/use_case_model_4C64E97D.html)

Representing relationships

There are several advanced modeling concepts available in the literature for structuring the use-case model, however, following the principle of "keep-it-simple" only the most useful of these, namely the <<include>> relationship is discussed in this process. This relationship permits one to factor out common behavior into a separate use case that is "include" in other use cases.

Another common type of relationship is the <<extend>> one. This may prove useful in dealing with exceptions in the normal use case flow of actions.

The generalization relationships may be used to improve the structure of the use case model.

Running each use case includes communication with one or more actors. A use-case instance is always started by an actor asking the system to do something. This implies that every use case must have communicates-associations with actors. The reason for this rule is to enforce that the system provides only the functionality that users need and nothing else. Having use cases that no one requests is an indication that something is wrong in the use-case model or in the requirements.

However, there are some exceptions to this rule:

An "included" use case might not interact with an actor if the base use case does.
A use case can be initiated according to a schedule (for example, once a week or once a day), which means that the system clock is the initiator. The system clock is internal to the system; therefore, the use case is not initiated by an actor but by an internal system event. If no other actor interaction occurs in the use case, it will not have any associations to actors. However, for clarity, you can use "time" as an actor to show how the use case is initiated in your use-case diagrams. CAUTION: if you have a lot of "time" actors in your model, challenge them. Perhaps you missed a real actor, such as an administrator responsible for scheduling reports, etc.

4.6.2. Reuse Guidelines

4.6.2.1. Composition

--

4.6.2.2. Dependency Relationship with other fragments

In most approaches, this fragment is intended to be the first of the design process but a requirements elicitation fragment can be adopted before this.
Most approaches for use case model definition requires the availability of scenarios as the basis for such a work.

4.7. References

1. Cossentino, M. 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
2. http://pa.icar.cnr.it/passi/

3. OpenUP website: http://epf.eclipse.org/wikis/openup/
4. Cossentino, M., Seidita, V. Metamodeling: Representing and Modeling System Knowledge in Design Processes. Technical Report ICAR-CNR n.11-02 (2011)
5. Seidita, V., Cossentino, M. and Chella, A. (2012). How to Extract Fragments from Agent Oriented Design Processes. *Proc. of the 11th International Conference on Autonomous Agents and Mutliagent System – The 13th International Workshop on Agent Oriented Software Engineering AOSE '12, to be held in Valencia (Spain) on June, 4th-5th, 2012.*
6. Seidita, V., Cossentino, M., Gaglio, S. (2006). A repository of fragments for agent systems design. *Proc. of the Workshop on Objects and Agents (WOA06).* Catania, Italy. 26-27 September 2006.
7. Seidita, V., Cossentino, M. and Chella, A. (2011). A Proposal of Process Fragment Definition and Documentation. *The 9th European Workshop on Multi-Agent Systems (EUMAS'11)* November 14th-15th, 2011, Maastricht (The Netherlands).
8.