Fragment Definition

Agents Identification

Version: 16 July 2008

Document Authors: M. Cossentino, V. Seidita

Index

1. Introduction

The PASSI process is composed of five different phases: System Requirements, Agent Society, Agent Implementation, Code and Deployment.
Each phase produces a document that is usually composed aggregating the UML models and work products of the work definitions that are inside each phase .

We will define a method fragment we call “Agent Identification”, extracted from PASSI methodology whose process is completely represented in the following figure

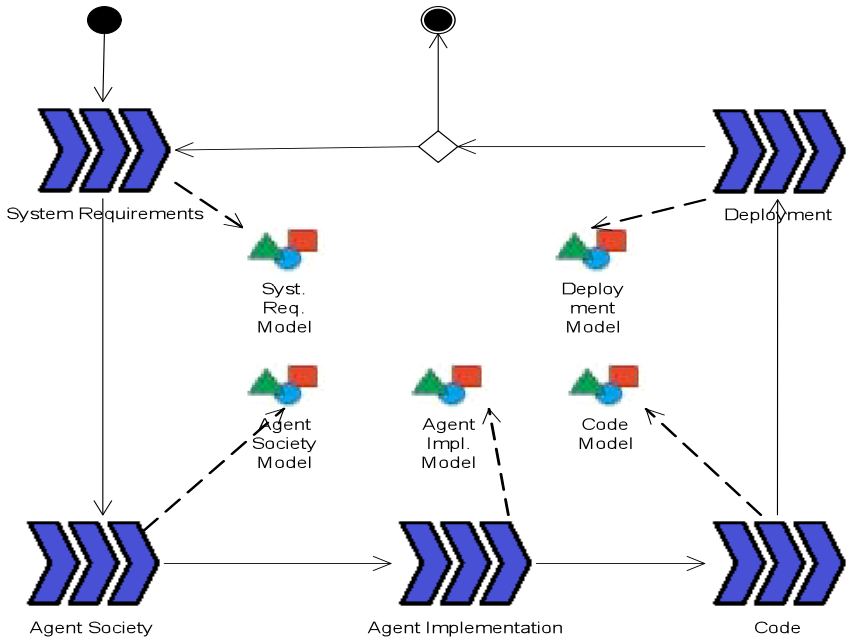


Fig. 1 The complete PASSI process

2. Fragment Description

The fragment here described is one of the peculiarities that distinguish the PASSI process from other approaches. The designer skill in capturing system requirements has been capitalized in order to produce an initial representation of the system functionalities (Domain Description Fragment) and now this model is used to identify agents and designate their responsibilities in terms of requirements to satisfy.

More in detail the System Requirements phase:

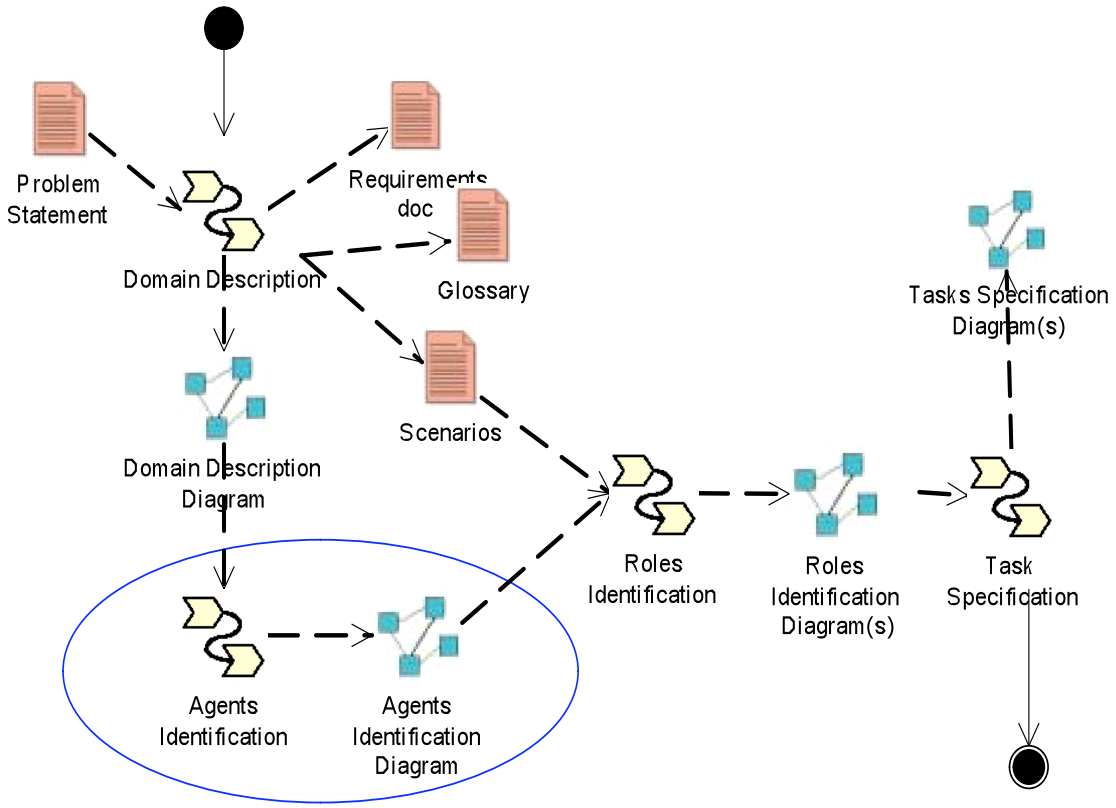


Fig.2 The System Requirements phase

Let us consider the "Agent Identification" sub-phase (the blue oval) .This fragment aims to identify all the agents involved in the system to be developed.

2.1. Portion of process

The process that is to be performed in order to obtain the result is represented in fig. 3 as a SPEM diagram

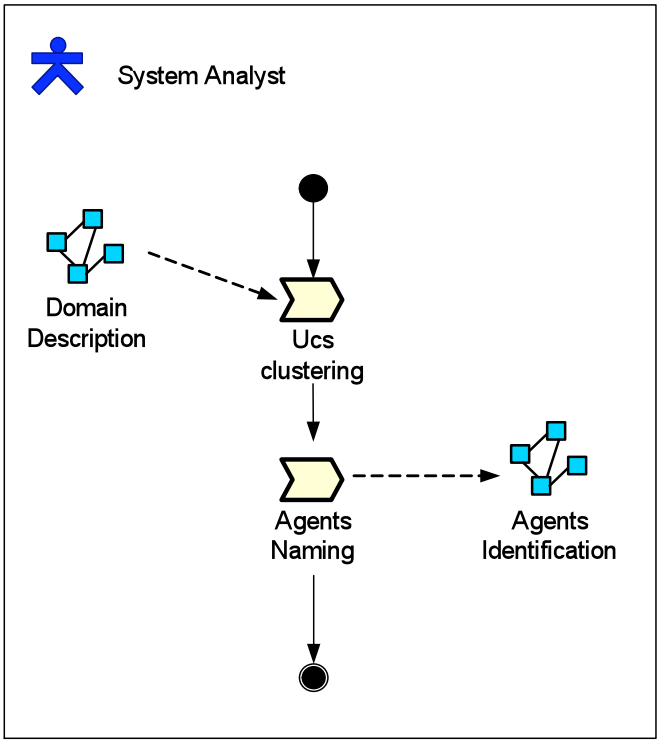


Fig 3 Agents Identification description fragment-Procedural aspect

Activities description:

Activity Name	Description	Roles involved
Use Cases Clustering	The System Analyst analyzes the use case diagrams resulting from the previous phase and attempts their clustering in a set of packages	System Analyst (perform)
Agents Naming	After grouping the use cases in a convenient set of packages, the last activity of this phase consists in identifying these packages with the names that will distinguish the different agents throughout all the project	System Analyst (perform)

System Analyst Role
In this fragment, he is responsible of performing all of the above described activities

3. Relationship with MAS meta-model

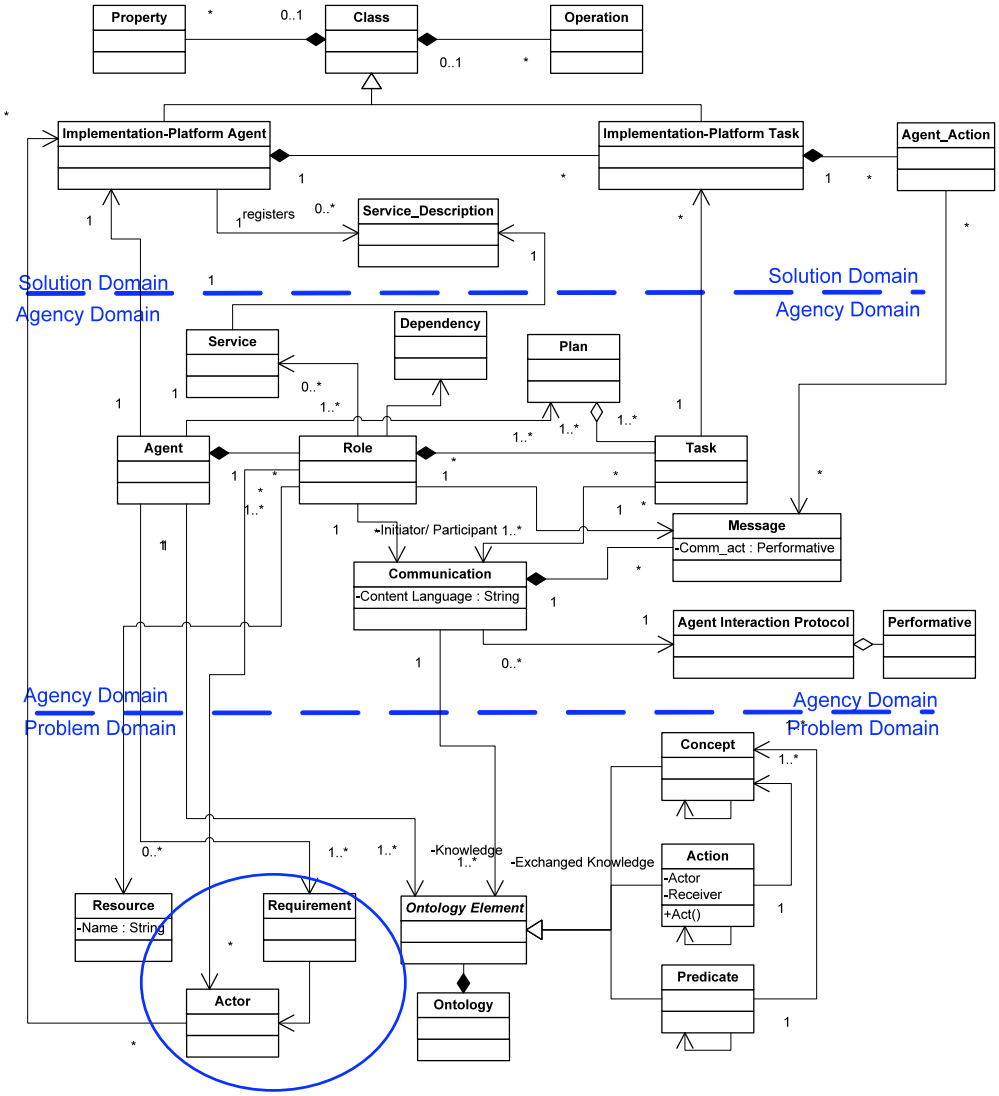


Fig5. The MAS meta-model adopted in PASSI

This fragment refers to the MAS meta-model adopted in PASSI and contributes to define the agent element of it.

4. Deliverables

The resulting artefact of this phase is an use case diagram (Agent Identification diagram) reporting the same use cases of the previous phase now clustered inside a set of packages, each one representing one agent. As it is common, we represent external entities interacting with our system (people, devices, conventional software systems) as actors.
Relationships between use cases of the same agent follow the usual UML syntax and stereotypes, whereas relationships between use cases of different agents are stereotyped as *communication* as described below.
Our assumptions about agent interaction and knowledge play an important role in the understanding of this phase and they are as follows:

- An agent acts to achieve its objectives on the basis of its local knowledge and capabilities;

- Each agent can request help from other agents that are collaborative if this is not in contrast with their own objectives;
- Interactions between agents and external actors consist of *communication* acts; this implies that if some kind of *include*/*extend* relationship exists between two use cases belonging to different agents, this stereotype is to be changed to *communication* since a conversation is the unique interaction way for agents. This is a necessary extension of the UML specifications that allow communication relationships only among use case and actors. The direction of the relationships goes from the initiator of the conversation to the participant. This stereotype change is, however, not in contrast with the spirit of the definition of the communication relationship since an agent is a proactive entity that could initiate an interaction just like an actor. An exception exists to this change in the relationship stereotype: it is possible that an agent in requiring some collaboration from another will not use a communication but instead will instantiate the other one; in this case, that is however not frequent, we use an *instantiate* stereotype to distinguish this situation from the others.
- An agent's knowledge can increase through communication with other agents or exploration of the real world.

Starting from an use case diagram, packages are used to group functionalities that will be assigned to an agent (whose name is the name of the package).

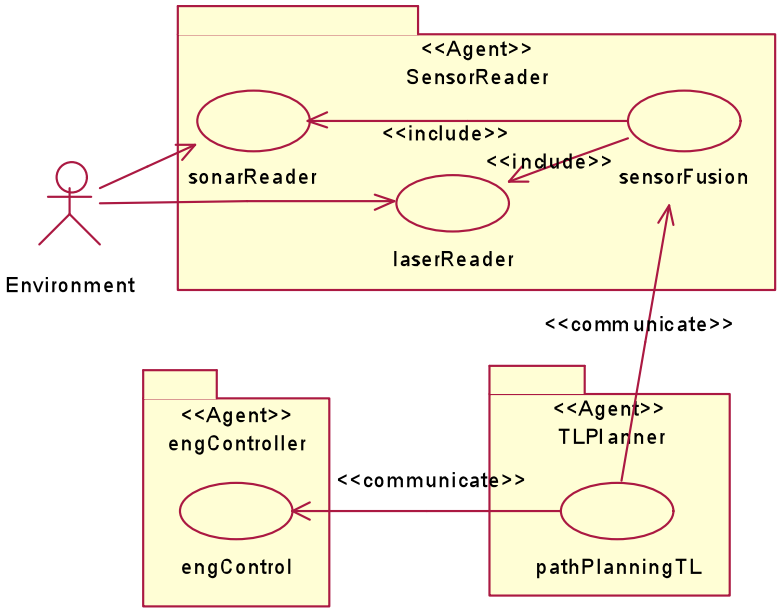


Fig. 4 The Agent Identification Diagram

5. Preconditions and concepts to be defined

Input, output and element to be designed in the fragment are detailed in the following table.

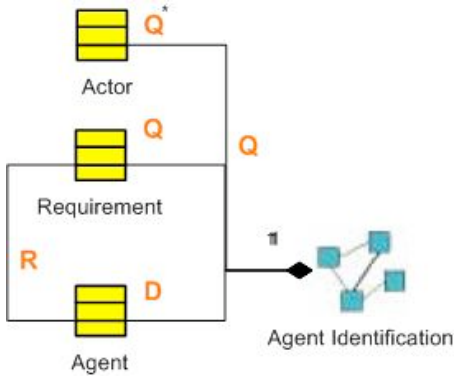
As regards documents:

Input	Output
Use Case diagram from the system requirements elicitation (Domain Description in PASSI)	Agent Identification (UML diagram)

As regards MAS metamodel elements:

To Be Designed	To be related	To be quoted
Agent	Agent-Requirement	Requirement

The following figure describes the structure of the work product produced in this fragment:



Note that the symbol:  *represents an element of the MAS model.*

The agent element is defined only by specifying its name and relationships with existing requirements.

6. Guideline

This phase is usually performed by a system analyst whose work is described in the SPEM activity diagram reported in Figure 3; the first activity consists in analyzing the use case diagrams resulting from the previous phase and attempt their clustering in a set of packages. Not precise rules exist to guide this operation but some guidelines could be drawn:

- It is better to group use cases that have inner logical commonalities because probably this will bring to implementations that have several common elements
- Data flow could represent an important problem for intrinsically distributed systems like MASs and therefore it could be useful to group together use case that will probably exchange a significant amount of data
- This activity produces a sort of architectural decomposition of the future system (at least at the functionality level but being each agent a consistent element of the implementation this partition also guides some kind of structural decomposition for the following solution). This suggests the observance of some common sense rules for agents identification:

- When possible (and if evident at this stage), agents that could be deployed in special devices (like PDA or cellular phones) should be fine grained in order to optimize their performance.
- Human interaction functionalities could be assigned to specific agents in order to prepare the option for a multi-device implementation (web-based, cell phone interfaces, and so on) via different categories of agents implementing these functionalities.
- In order to facilitate agents mobility, functionalities that strictly depend on hardware devices or databases should that could not be accessed by everywhere should be divided by the remaining part of the system eventually using a wrapping solution.

7. Composition Guideline

The fragment can be used after a functional-oriented requirements elicitation (performed with use case diagrams) in order to identify a system decomposition into agents. It is not good for goal-oriented approaches.

8. Aspects of Fragment

Behind this fragment there is only the basic assumption that the system is to be modelled in terms of (functional) requirements.

9. Dependency Relationships with other fragments

None specific, obviously as already discussed in section 7 and 5, an use case diagram representing system requirements is necessary as an input.

10. Glossary

Agent Identification Fragment uses this list of model elements:

Agent – an autonomous entity that is composed by roles and has a knowledge. An agent can be seen from different level of abstraction. In this fragment agents are a logical aggregation of functionalities (Use Case diagrams).
In general in PASSI, an agent is a significant software unit at both the abstract and concrete levels of design. According to this view, an agent is an instance of an agent class. So it is the software implementation of an autonomous entity capable of going after an objective through its autonomous decisions, actions and social relationships. An agent may undertake several functional roles during interactions with other agents to achieve its goals. A role is a collection of tasks performed by the agent in pursuing a sub-goal. A task, in turn, is defined as a purposeful unit of individual or interactive behaviour.
Requirement - A requirement represents a feature that the system to be must exhibit, it can be a functional requirement that describes the interactions between the system and its environment independent of its implementation, or a non-functional requirement such as a constraint on the system (or a specific part of it) performance.