Agents Identification

Process Fragment

Author(s): M. Cossentino, V. Seidita

Last saved on: 04/11/10 16:24

Index

Fragment Description

Fragment Goal

Assigning system functionalities to the responsibility of newly defined agents

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.

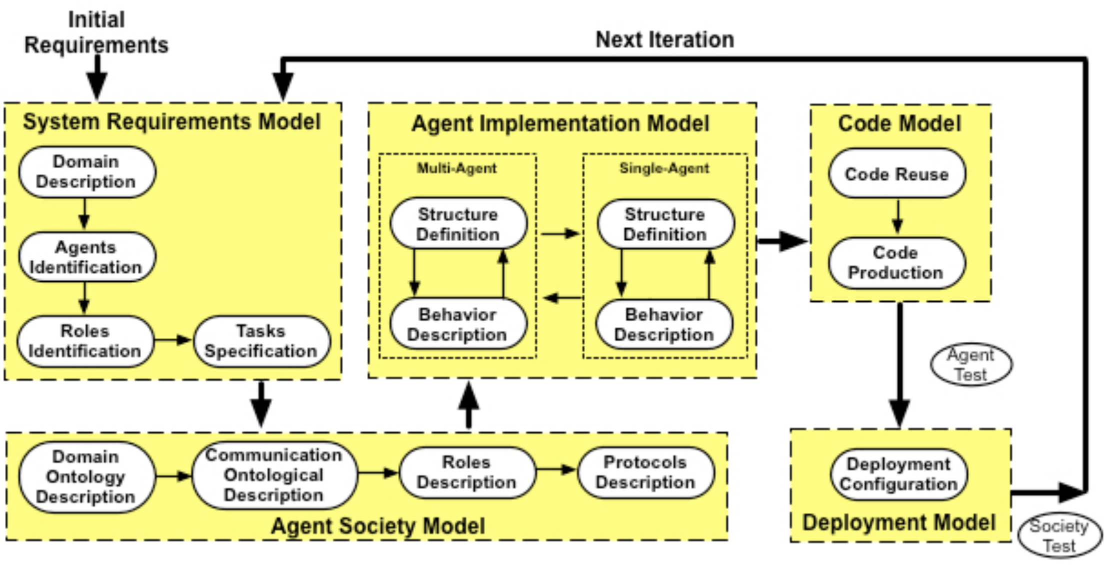

Figure 1. The PASSI design process

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.

The PASSI Process lifecycle

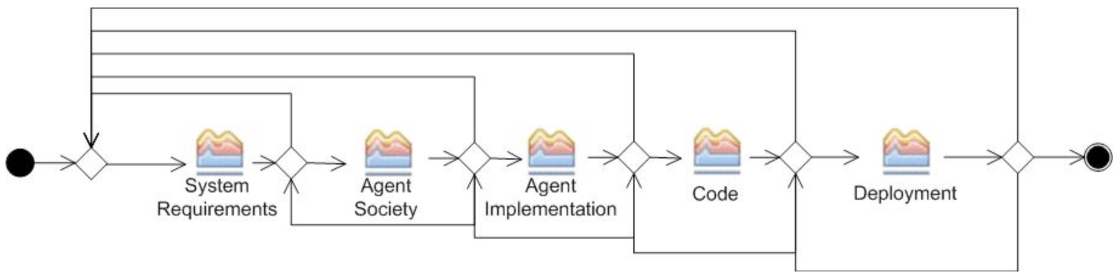

Figure 2. The PASSI process phases

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.

Fragment Overview

The fragment here described is one of the peculiarities distinguishing 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 (PASSI Domain Description Fragment) and now this model is used to identify agents and designate their responsibilities in terms of requirements to be satisfied.
Let us consider the "Agent Identification" sub-phase (red box in Figure 3). This fragment aims to identify all the agents involved in the system to be developed.

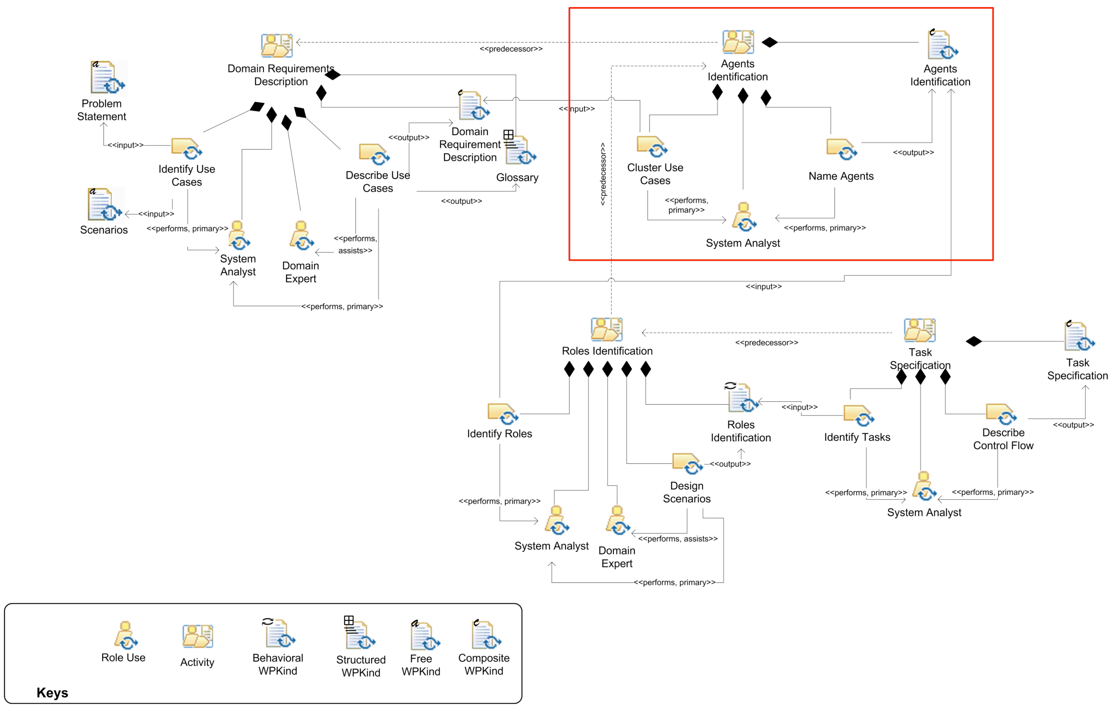

Figure 3. ***The System Requirements Phase (structural view)***

Fragment System metamodel

The portion of metamodel of this fragment is:

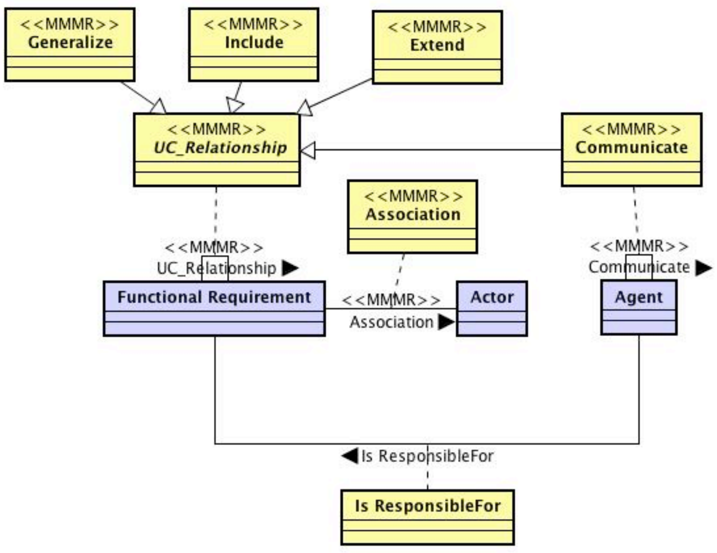

Figure 4. The fragment system metamodel

This fragment refers to the MAS metamodel adopted in PASSI and contributes to define and describe the elements reported in Figure 3.

Definition of System metamodel elements

This fragment underpins the following model elements:

Agent – the system requirements domain agent is a responsibility center; this means that each agent will rationally act to achieve its objectives (usually defined in terms of functionalities it should ensure).
Generally speaking, an Agent is an entity which:
- is capable of actions in an environment;
- can communicate directly with other agents;
- 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;
- can play several, different (and sometimes concurrent or mutually exclusive) roles.

Functional requirement - Functional requirements describe the functions that the software is to execute. (from IEEE SEBOK 2004)

Actor - An external entity (human or system) interacting with the multi-agent system.

Definition of System metamodel relationships

Generalize – (standard UML meaning)
Include – (standard UML meaning)
Extend – (standard UML meaning)
Association – (standard UML meaning)
Communicate

System metamodel Input/Output

Input, output system metamodel elements to be designed in the fragment are detailed in the following tables.

As regards system metamodel elements:

Input		**To Be Designed**		**To Be Refined**		**To Be Quoted**	
MMME	MMMR	MMME	MMMR	MMME	MMMR	MMME	MMMR
Non Functional Requirement		Agent	Agent - Functional Requirement (Is Responsible for)		Functional Requirement – Functional Requirement (Generalize or Extend or Include)*	Functional Requirement	Actor-Functional Requirement
Functional Requirement						Actor	Functional Requirement – Functional Requirement (UC_Relationship)

Actor							

* Relationships connecting requirements now belonging to different agents are changed to Communicate type.

Definition of input system metamodel elements and relationships

Non-Functional requirement - Non functional requirements constrain the solution and are sometimes known as constraints or quality requirements. (from IEEE SEBOK 2004)

Stakeholder

Roles involved in this fragment are:

- System Analyst

Their responsibilities are described in the following subsections.

System Analyst

He is responsible for:

1. Use Cases Clustering. The System Analyst analyzes the input use case diagrams and define their clustering in a set of packages.
2. Agents Naming. 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 the project.

Fragment workflow

Workflow description

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

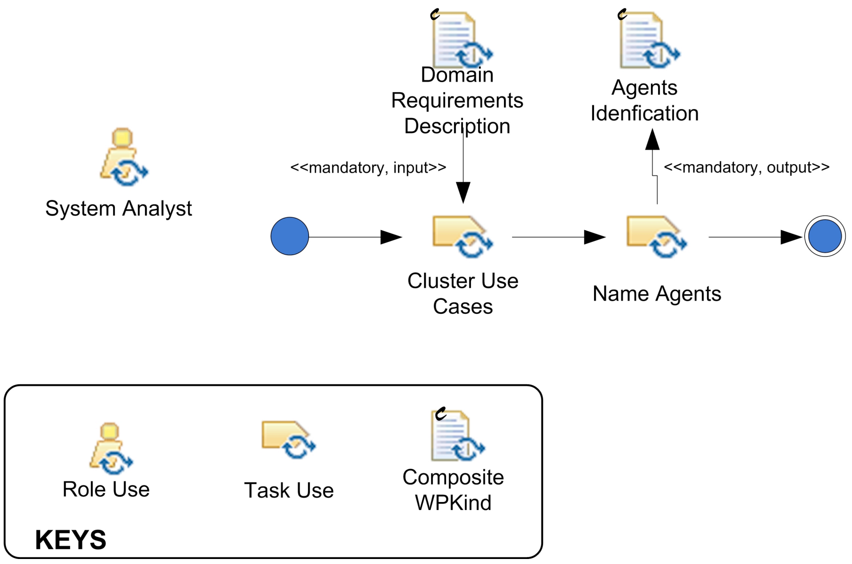

Figure 5. The flow of activity of this fragment

Activity description

The fragment encompasses the following *work breakdown elements*:

Name	Kind	Description	Roles involved
Use Cases Clustering	Task	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	Task	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 metamodel elements and relationships input/output

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

	Input		Output	
Activity/Task Name	MMME	MMMR	MMME	MMMR

Use Cases Clustering	Actor, Functional Requirement, Non Functional Requirement.	Functional Requirement -Functional Requirement (Generalize, Include, Extend)		
		Functional Requirement -Actor (Association)		
		Functional Requirement - Non Functional Requirement (Constrained By)		
Agents Naming			Agent	Communicate

WP Input/Output

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

Input	**Output**
Domain Requirements Description document	Agents Identification Document

Deliverable

Agents Identification document

The resulting artefact of this phase is a composite document composed by:

- an use case diagram (Agent Identification diagram) reporting use cases now clustered inside a set of packages, each one representing one agent.
- a table describing agents' main features (requirements, constraints, …)

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.

Example of notation

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).
Other details about the adopted notation descend from the discussion reported in other subsections (for instance introduction of communicate relationships among agents).

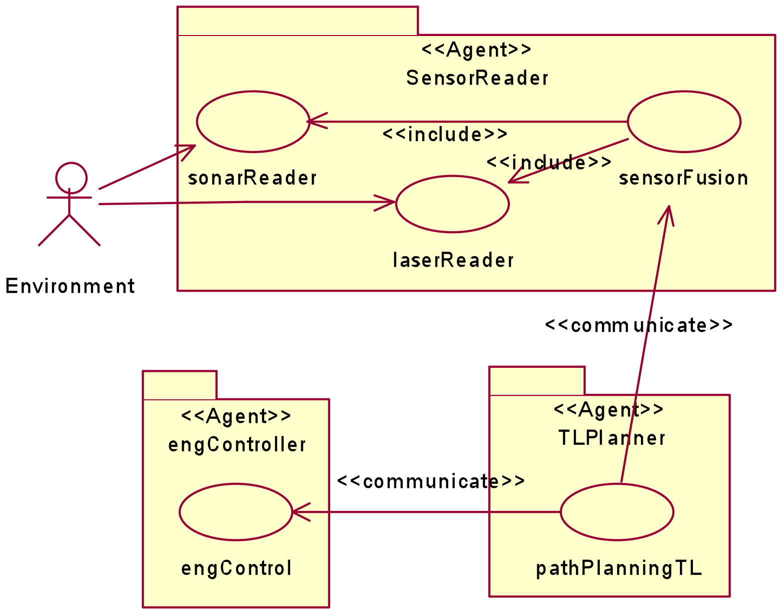

Figure 6. An example of Agent Identification Diagram

The following table is used to describe agents' main features. The table may be composed by reusing information from Domain Requirements Description document in the PASSI process.

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

Deliverable relationships with the MMM

The following figure describes the structure of this fragment work products in relationship with the MAS model elements:

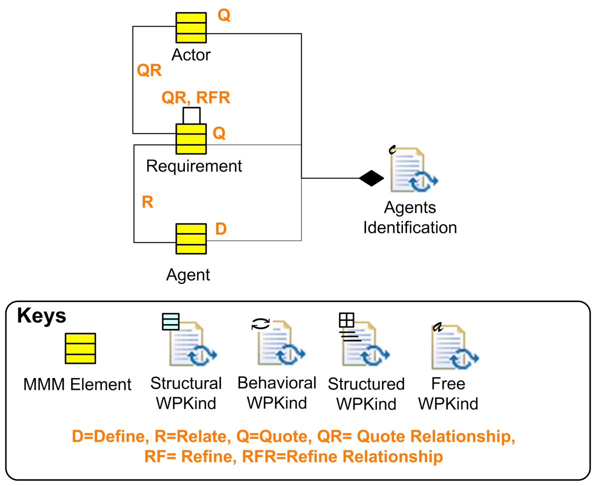

Figure 7. Structure of the fragment work-product in terms of MAS meta-model elements

Guidelines

Enactment Guidelines

Some assumptions about agent interaction and knowledge play an important role in the understanding of this activity 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.

Functional requirements (represented by use cases) can be clustered according to different criteria that can be adopted by the designer in different situations:

- Use cases can be clustered according to functional similarity or dependency. This brings to agents with a high functional coherence.
- Use cases can be clustered according to similarities in information they manage or exchange. This has a positive effect on information management also in terms of security (if needed).

- Use cases can be clustered according to involved actors. This allows the optimisation of graphical interfaces since all the interactions involving the same actor are managed by the same agent (or a small set of agents).
- 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:
    a. 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.
    b. 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.
    c. 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.

Reuse Guidelines

Composition

The only input for this fragment is a system requirement description; as a consequence, this fragment can be used after a functional-oriented requirements elicitation (performed with use case diagrams) in order to identify the system decomposition into agents. It is not suggested for goal-oriented approaches.

Dependency Relationship with other fragments

This fragment is conceived to receive a system requirements document like the work product produced by the PASSI Domain Requirements Description fragment.

References

1. 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
2. http://pa.icar.cnr.it/passi/