Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

DRAFT

SPEM 2.0 description of the PASSI process

Massimo Cossentino, Valeria Seidita

RT-ICAR-	Date: 26/01/10 17:24

Consiglio Nazionale delle Ricerche, Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR) – Sede di Cosenza, Via P. Bucci 41C, 87036 Rende, Italy, URL: *www.icar.cnr.it*
– Sezione di Napoli, Via P. Castellino 111, 80131 Napoli, URL: *www.na.icar.cnr.it*
– Sezione di Palermo, Viale delle Scienze, 90128 Palermo, URL: *www.pa.icar.cnr.it*

Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

DRAFT

SPEM 2.0 description of the PASSI process

Massimo Cossentino[1], Valeria Seidita[2]

Rapporto Tecnico N.:	*Data:*
RT-ICAR-	**26/01/10 17:24**

[1] Istituto di Calcolo e Reti ad Alte Prestazioni, ICAR-CNR, Sezione di Palermo, Viale delle Scienze edificio 11, 90128 Palermo (Italy).

[2] Dipartimento di Ingegneria Informatica, Università degli Studi di Palermo, Viale delle Scienze edificio 6, 90128 Palermo (Italy).

I rapporti tecnici dell'ICAR-CNR sono pubblicati dall'Istituto di Calcolo e Reti ad Alte Prestazioni del Consiglio Nazionale delle Ricerche. Tali rapporti, approntati sotto l'esclusiva responsabilità scientifica degli autori, descrivono attività di ricerca del personale e dei collaboratori dell'ICAR, in alcuni casi in un formato preliminare prima della pubblicazione definitiva in altra sede.

Index

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, 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 thus encouraging the use of this process even in time/cost-constrained projects or where high quality standards are requested.

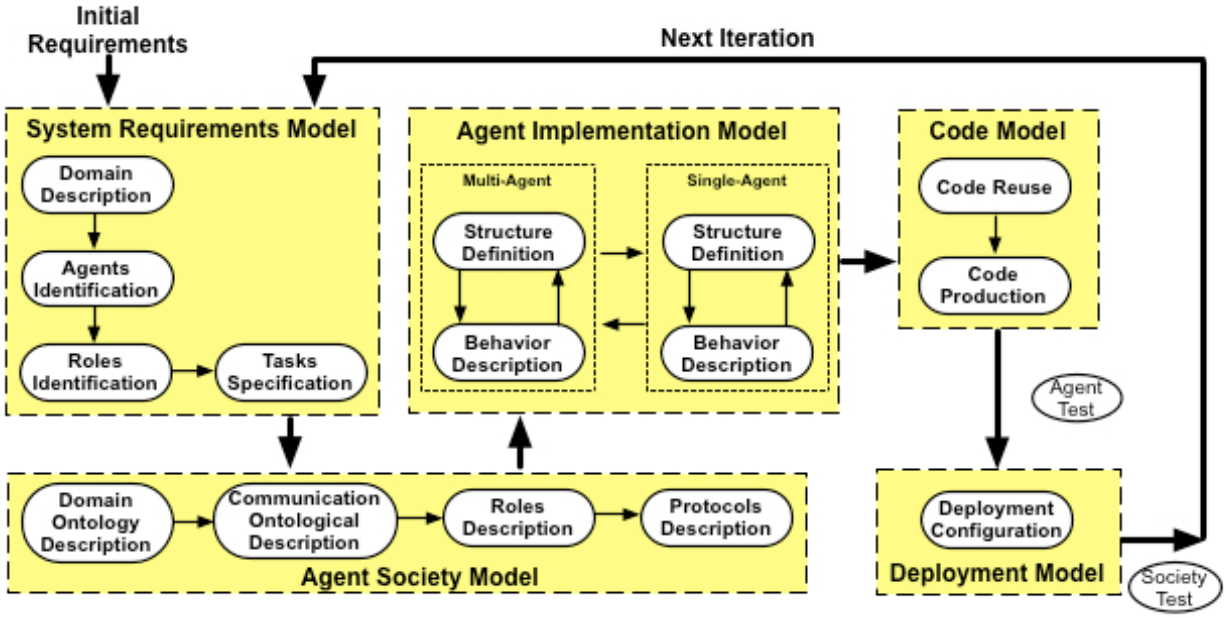


Figure 1. The PASSI design process

The design process is composed of five models (see Fig. 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 (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:

- 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:

- 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.

The PASSI Process lifecycle

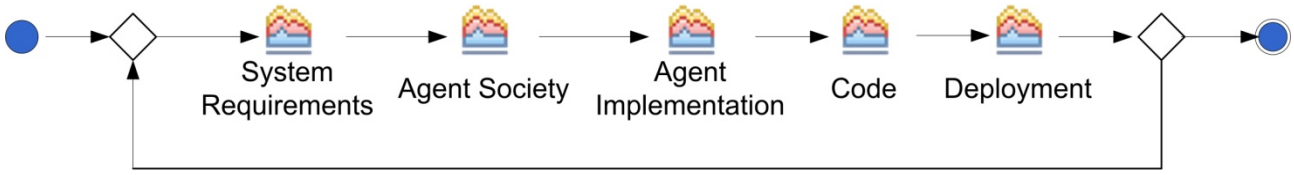


Figure 2. The PASSI process phases

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

- 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).
The details of each phase will be discussed in the following section.

The PASSI MAS Metamodel

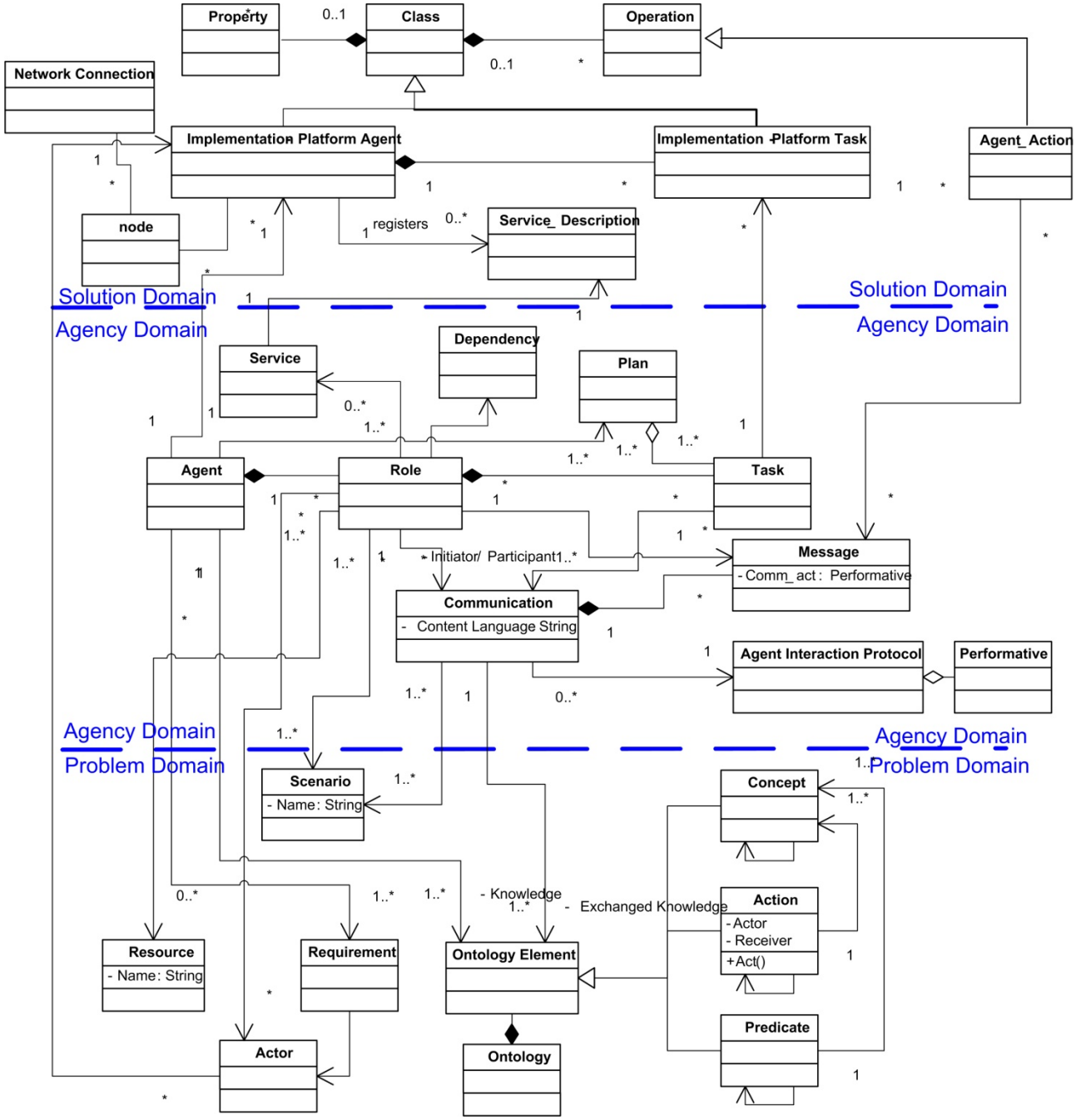


Figure 3: The PASSI metamodel

The description of the PASSI MAS meta-model (Figure 3) addresses three logical areas: (i) the problem domain, (ii) the agency domain, and (ii) the solution domain; they are introduced in an order that reflects our choice of an agent approach for solution refinement and modelling that is based on Model Driven Engineering.

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

Then we introduce the agency domain components; they 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 (we suppose 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, we can see that (Figure 3), 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.
In PASSI an agent is responsible for realizing some functionalities descending from one or more functional requirements. It also has to respect some non functional requirement constraints (like for instance performance prescriptions). It 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.
Each agent during its life plays some roles. A role is a peculiarity of the social behavior of an agent. When playing a role, an agent may provide a service to other agents.
The role concept is tightly related to the agent communication capability (interactions among agents are uniquely based on communications composed of messages seen as speech acts).
In PASSI, a task specifies the computation that generates the effects of a specific agent behavioral feature. It is used with the significance of atomic part for defining the overall agent's behavior. This means that an agent's behavior can be composed by assembling its tasks and the list of actions that are executed within each task cannot be influenced by the behavior planning. Tasks are structural internal components of an agent and they contribute to define the agent's abilities; these cannot be directly accessed by other agents (autonomy) unless the agent offers them as a set of services.
A communication is an interaction among two agents and it is composed of one or more messages. The information exchanged during a communication is composed of concepts, predicates or actions defined in the ontology. The flow of messages and the semantic of each message are ruled by an agent interaction protocol (AIP).
The last Agency Domain element (Service) 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 and 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 file) of each service specified in the Agent Domain.

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 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 which: - 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
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 Ontologies therefore provide 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

Phases of the PASSI Process

The System Requirements Phase

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 4.

The phase is composed of four activities (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).

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 sub-sections (after the description of process roles).

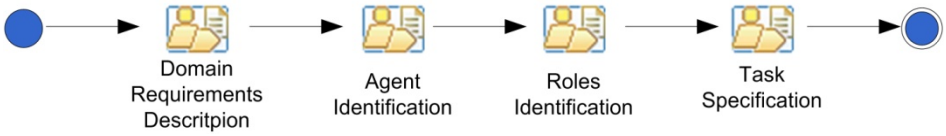


Figure 4. The System Requirements Phase flow of activities.

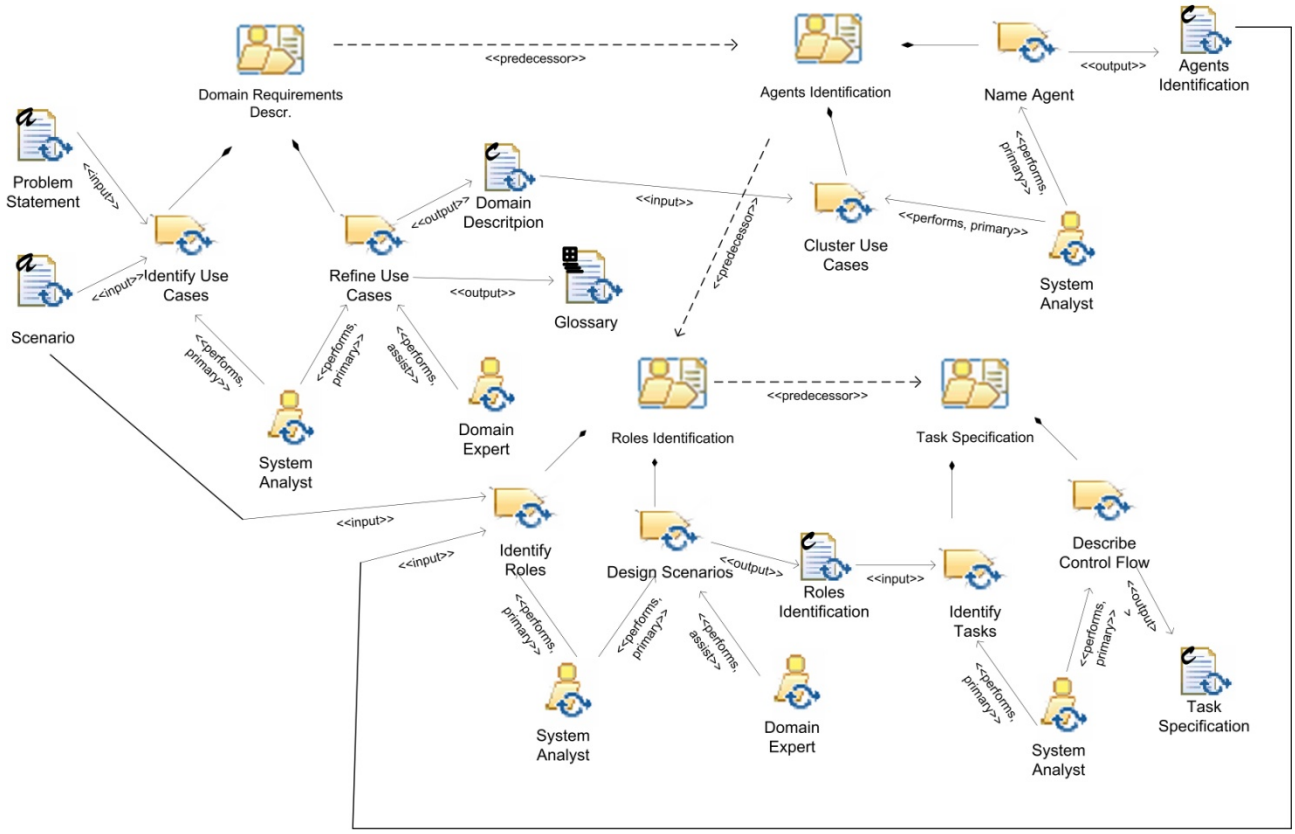


Figure 5. The System Requirement phase described in terms of activities and work products

Process roles

Two roles are involved in the System Requirements discipline: the System analyst and the Domain Expert. They are described in the following sub-sections.

System Analyst

He is responsible of:

1. Use cases identification during the DD sub-phase. Use cases are used to represent system requirements.
2. Use cases refinement during the DD sub-phase. Use cases are refined with the help of a Domain Expert.
3. Use cases clustering during the AId sub-phase. The System Analyst analyzes the use case diagrams resulting from the previous phase and attempts their clustering in a set of packages.
4. Naming agents during the AId sub-phase. 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 RId sub-phase. 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 sub-phase. Each scenario in designed in form of sequence diagram thus depicting the details of agents interactions
7. Tasks identification during the TSp. sub-phase. It consists in the identification of the behavioral capabilities that each agent needs to perform the specified roles and the fulfill the requirements that are under its responsibility.
8. Description of the control flow during the TSp. sub-phase. It consists in introducing the communication relationships among tasks of different agents and the control flow among tasks of the same agent.

Domain Expert

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

Activity Details

Domain Requirements Description

The flow of tasks inside this activity is reported in Figure 6 and the tasks are detailed in the following table.

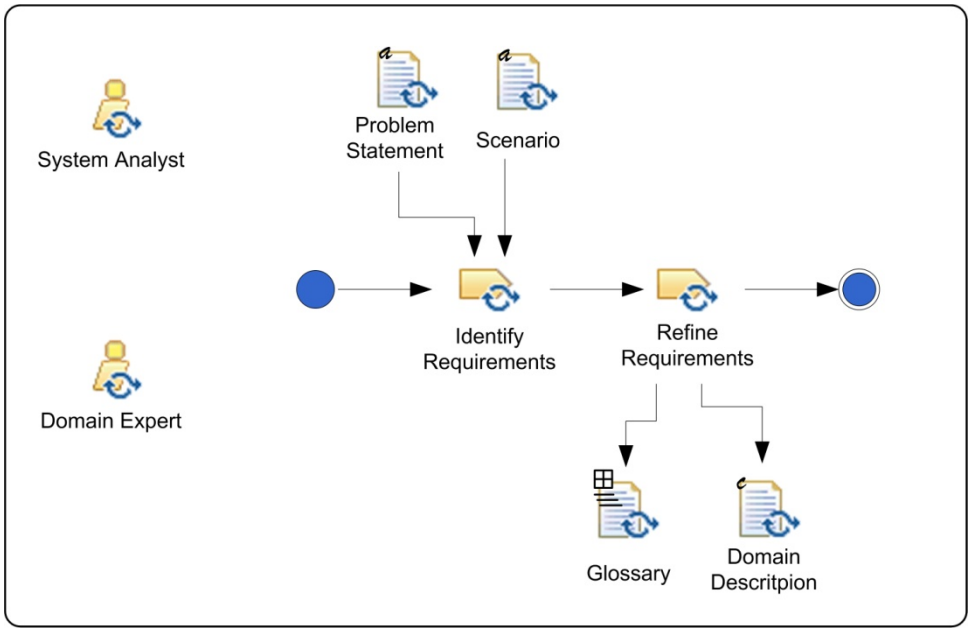


Figure 6: The flow of tasks of the Domain Requirements Description activity

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)

Agent Identification

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

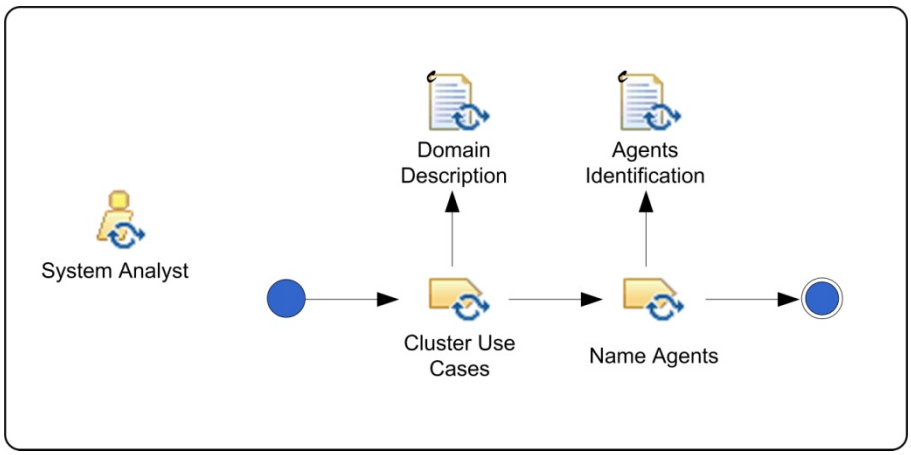


Figure 7: The flow of tasks of the Agent Identification activity

Activity	Task	Task Description	Roles involved
Agents Identification	Use Cases Clustering	The System Analyst analyses the use case diagrams resulting from the previous phase and attempts their clustering in a set of package.	System Analyst (perform)

Agents Identification	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)

Roles Identification

The flow of tasks inside this activity is reported in Figure 8 and the tasks are detailed in the following table.

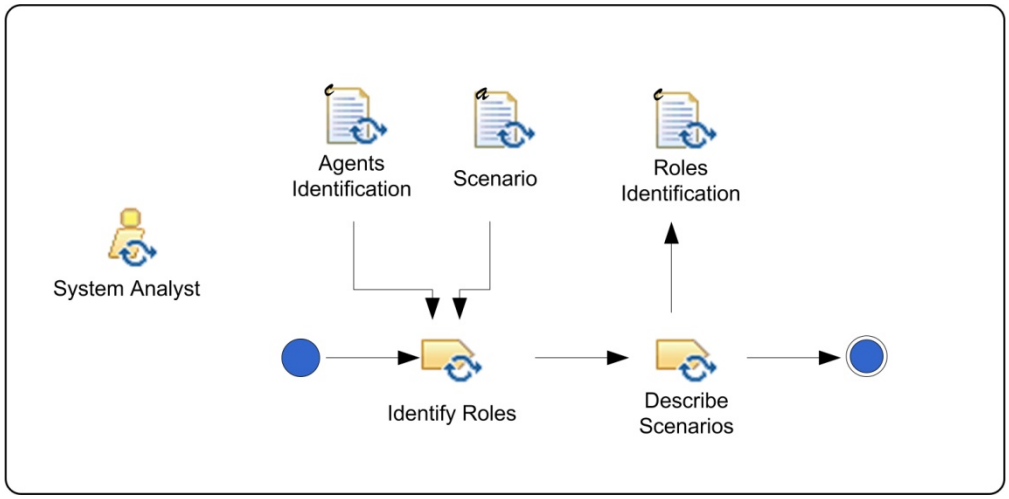


Figure 8: The flow of tasks of the Roles Identification activity

Activity	Task	Task Description	Roles involved
Role Identification	Identify Role	The System Analyst studies (textual) scenarios and system requirements (as defined in the previous phase) and identifies the roles played by agents	System Analyst (perform)
Role Identification	Design Scenarios	Each scenario in designed in form of sequence diagram thus depicting the details of agents interactions	System Analyst (perform) Domain Expert (assist)

Task Specification

The flow of tasks inside this activity is reported in Figure 9 and the tasks are detailed in the following table.

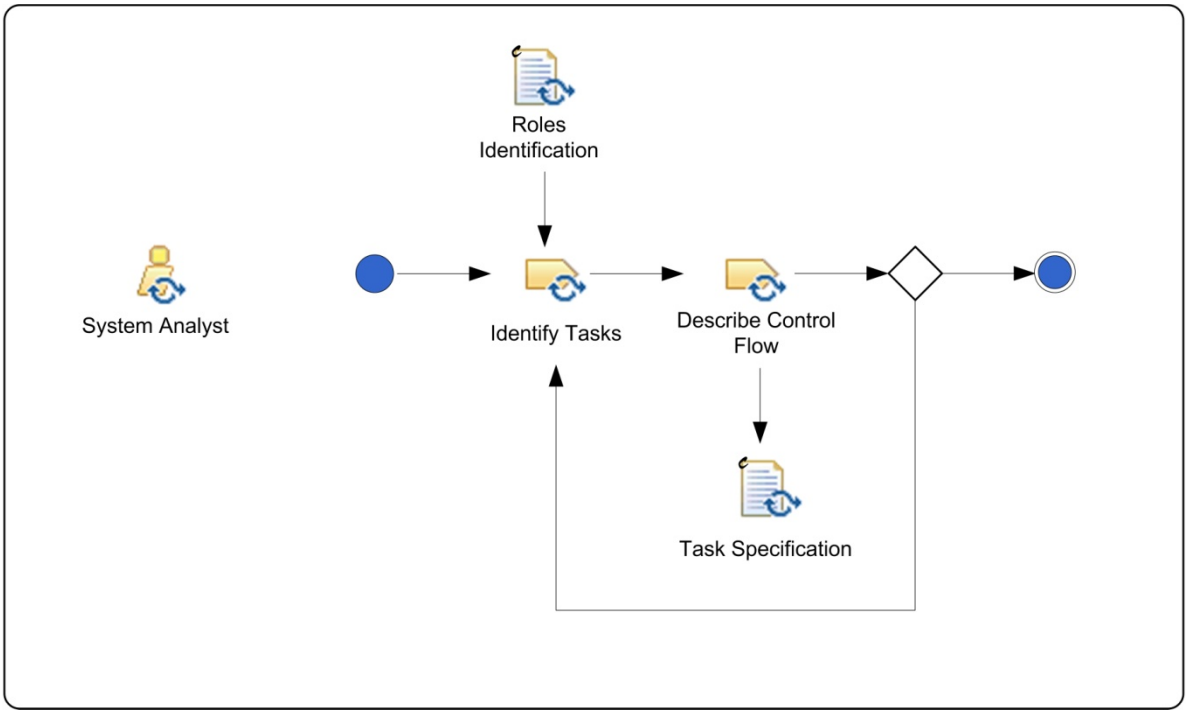


Figure 9: The flow of tasks of the Task Specification Activity

Activity	Task	Task Description	Roles involved
Task Specification	Identify Tasks	It consists in the identification of the behavioral capabilities that each agent needs to perform the specified roles and the fulfill the requirements that are under its responsibility	System Analyst (perform)
	Describe the control flow	It consists in introducing the communication relationships among tasks of different agents and the control flow among tasks of the same agent	

Work Products

The System Requirements Model generates four composed work products (text documents including diagrams). Their relationships with the MAS meta-model elements are described in following Figure 10.

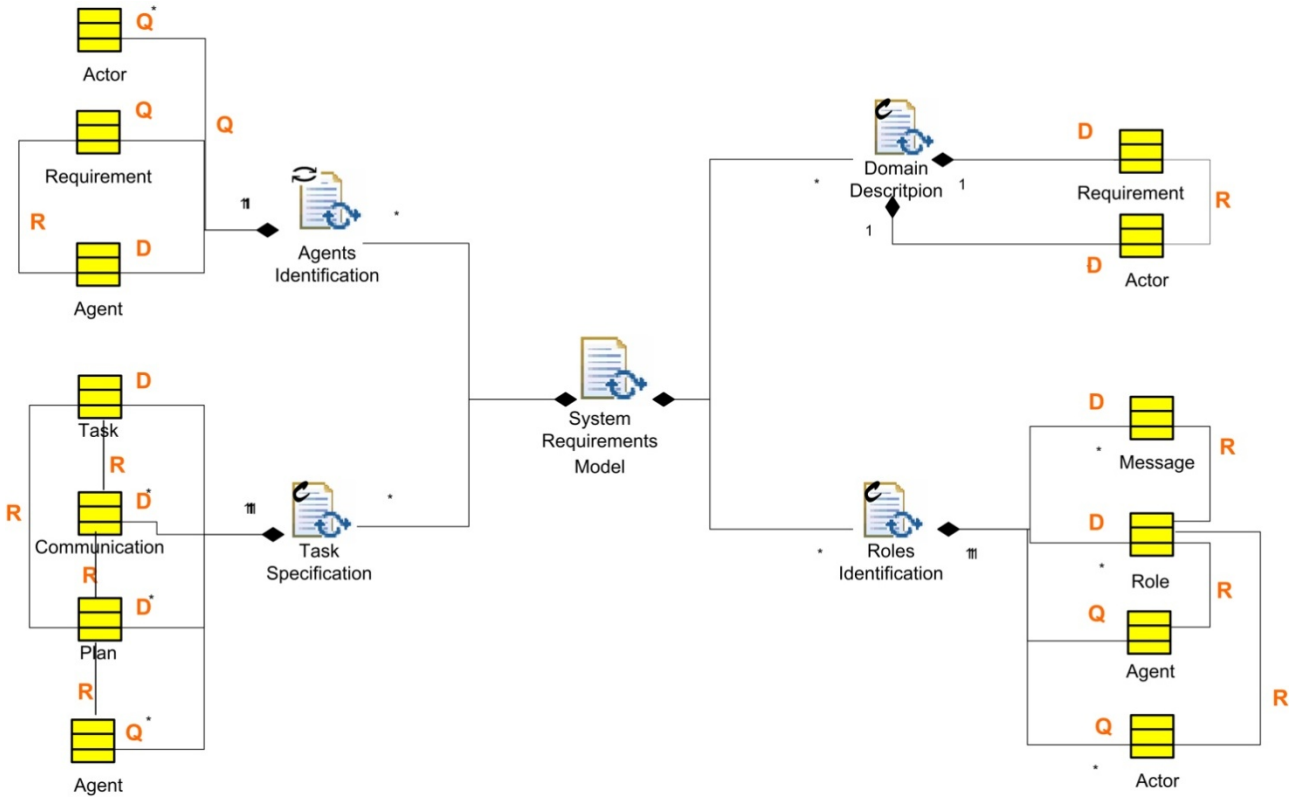


Figure 10. The System Requirements Phase documents structure

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 as described below:

- *defined:* D label near the element symbol, this means that the element is introduced for the first time in the design in this artefact (the MAS metamodel element is instantiated in this artefact).
- reFined: F label means that the MAS metamodel element is refined in the work product (for instance by means of attribute definition).
- *related:* R label, this means that an already defined element is related to another or from a different point of view that one of the MAS metamodel relationships is instantiated in the document).
- *quoted:* Q label, this means that the element has been already defined and it is reported in this artefact only to complete its structure but no work has to be done on it.
- Relationship Quoted: RQ label means that the relationship is reported in the work product but it has been defined in another part of the process.

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

Domain Requirements Description

Common UML use case diagram(s) are used to represent a functional description of the system. Use cases documentation is inspired by the approach proposed in: I. Jacobson, M. Christerson, P. Jonsson, and G. Overgaard. Object-oriented software engineering: A use case driven appraoch. Reading, MA: Addison Wesley, 1992.

Identifier: Unique identifier of the use case, may depend on reference number and modification history.
Name: Name of the use case.
Iteration: informs the reader of the stage the use case has reached.
Type: *Abstract* or *Concrete*
Description: Goal to be achieved by use case and sources of requirements.
Actors: List of actors involved in use case.
Pre-conditions: Conditions that must be true for use case to start successfully.
Post-conditions: Conditions that must be true at the end of the scenario. It summarizes the state of affairs after the scenario is complete.
Flow: The ordered list of interactions between actors and system that are necessary to achieve goal. Example of general flow of interaction:

1. Interaction x
2. Interaction y
3.a If condition then
    3.a.1 Interaction
    3.a.2 Interaction
3.b else If condition then
    3.b.1 Interaction
    3.b.2 Interaction
3.c else
    3.c.1 Interaction

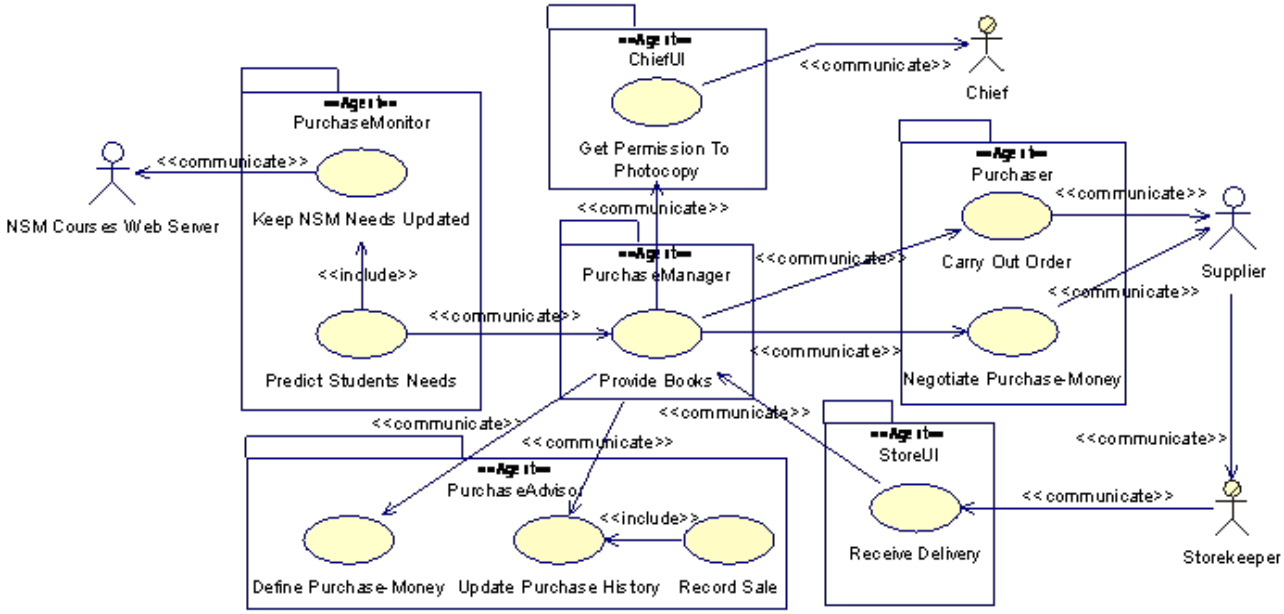
⋮

Variations/Alternative Flow: Any variations, alternative in the flow of a use case.
Non-functional: List of non-functional requirements that the use case must meet.
Authors and dates: Modifications history.

An example of documentation is reported in the following figure and table

Actor Name	Description

Agent Identification



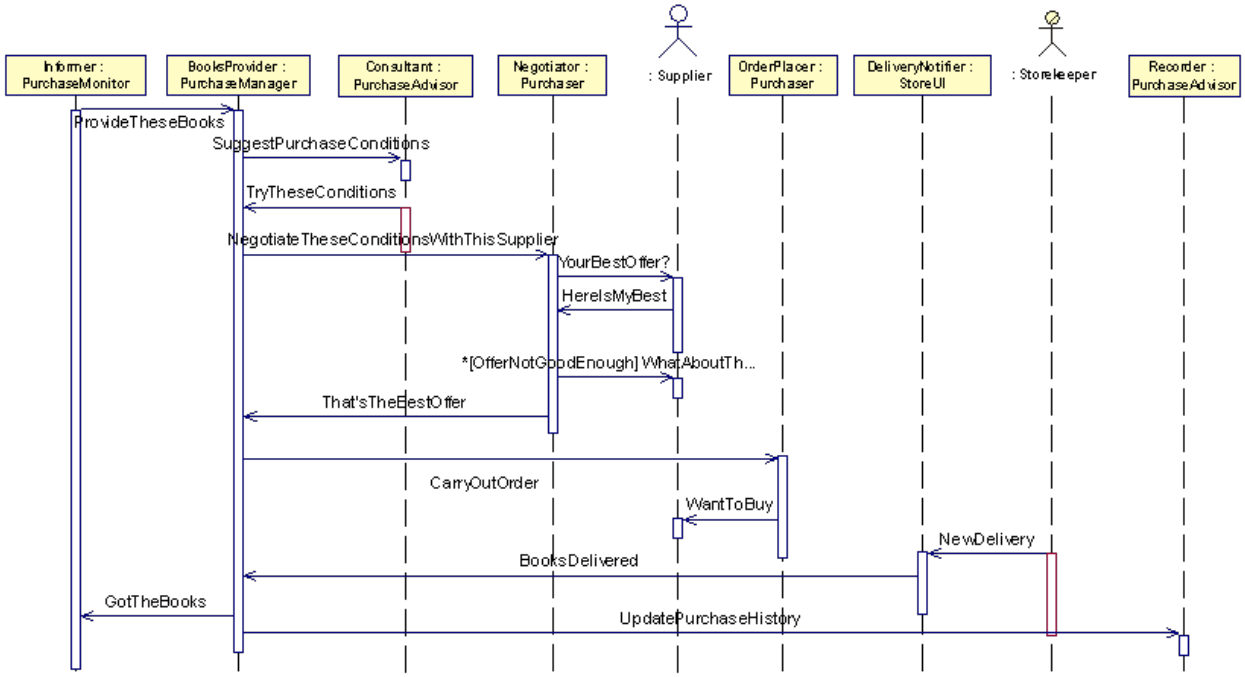
Starting from an use case diagram, packages are used to group functionalities that will be assigned to an agent (whose name is the name of the package).
Stereotypes of relationships between use cases of different packages (agents) are converted to 'communicate' since different agents can interact only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

The diagram is completed by the following information:

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

Roles Identification

During this phase all the possible communication paths between agents are represented. A path describes a scenario of interacting agents working to achieve a required behaviour of the system. Each agent may belong to several scenarios, which are drawn by means of sequence diagrams in which objects are used to represent roles (collection of tasks performed by agent in pursuing a sub-goal).

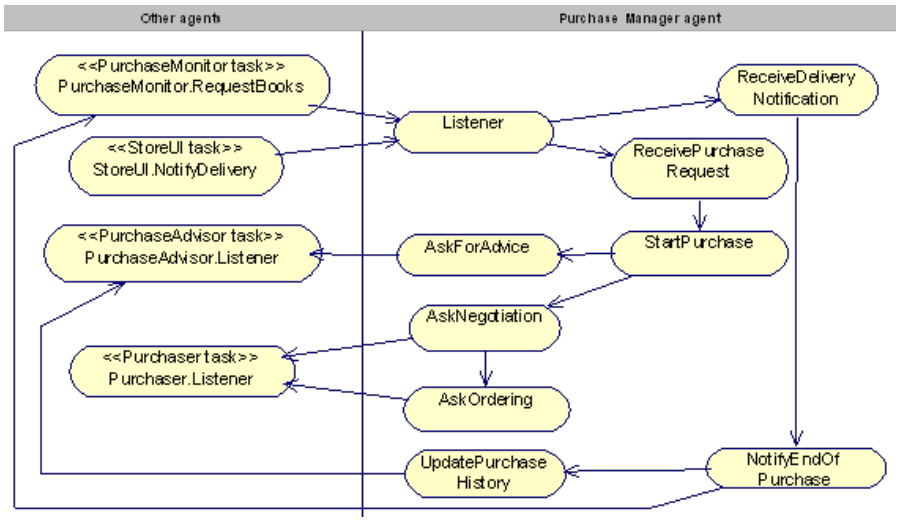


Each diagram is completed by the following information:

Role Name	Agent which plays it	Description	Responsibilities

Task Specification

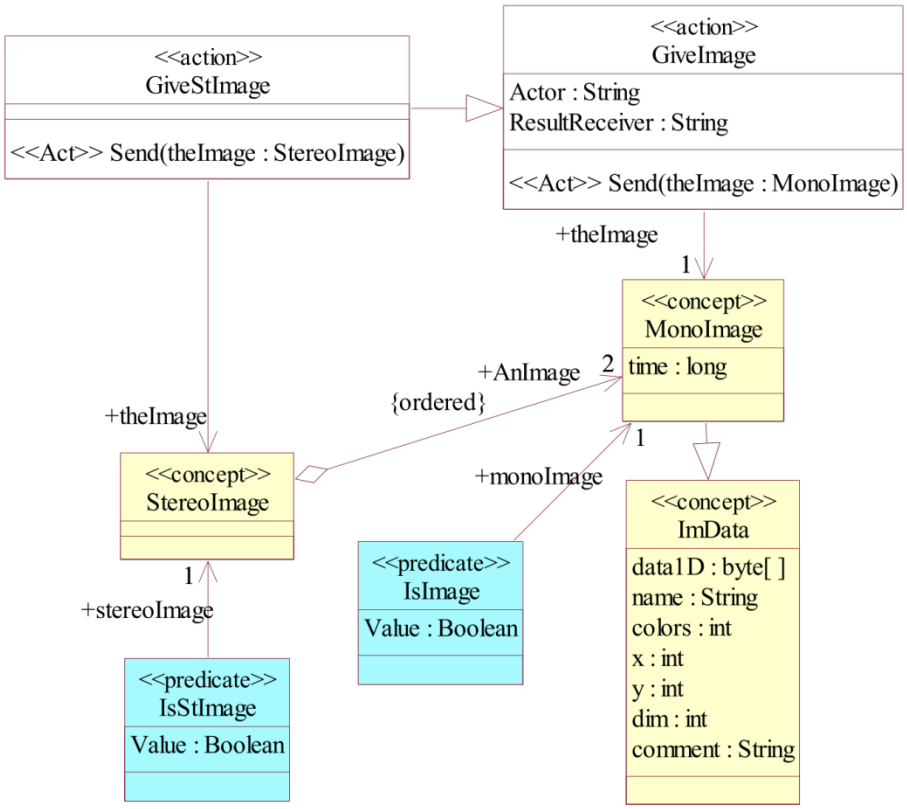
Activity diagrams are used to show the behaviour of each agent pointing attention in what it is capable to do. Relationships between activities signify messages and communications between tasks of the same agent. A Task specification diagram represents the plan of the agent behavior. It shows the relationships among the external stimuli received by the agent and its behaviour (expressed in terms of fired tasks).



Each diagram is completed by the following information:

Task name	
Description	
Actions	
Data	
Behavior	

Domain Ontology Description



The ontology is described (using a class diagram) in terms of concepts (fill colour : yellow), predicates (fill colour: light blue) and actions (fill colour: white).
Elements of the ontology can be related using three UML standard relationships:

1 Generalization: it permits the generalize/specialization relation between two entities that is one of the fundamental operator for constructing an ontology.

2 Association: it models the existence of some kind of logical relationship between two entities. It is possible to specify the role of the involved entities in order to clarify the structure.

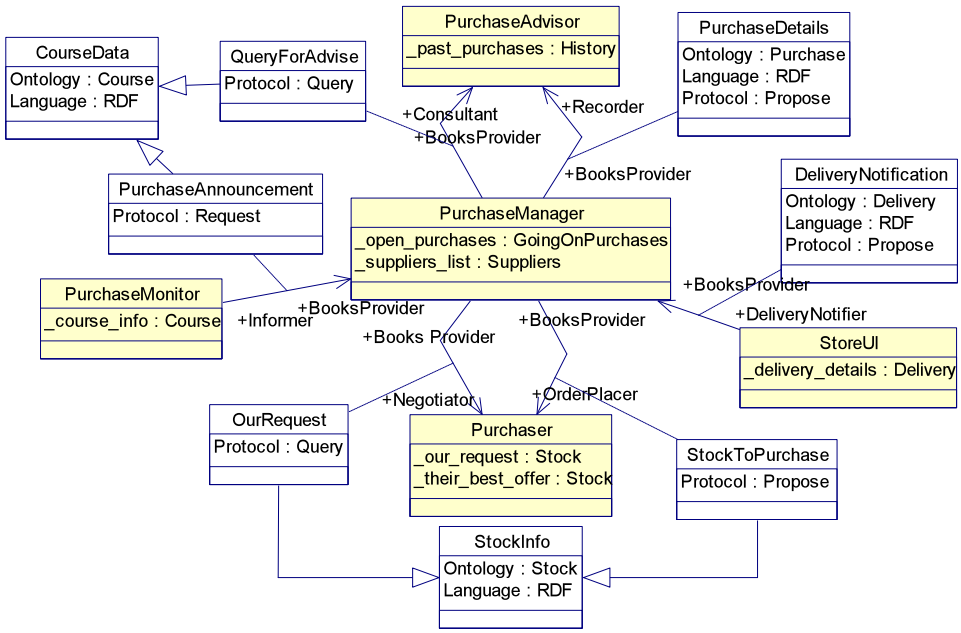
3 Aggregation: it can be used to construct sets where value restrictions can be explicitly specified; in the W3C RDF standard three types of container objects are enumerated: the bag (an unordered list of resources), the sequence (an ordered list of resources) and the

alternative (a list of alternative values of a property). We choose of considering a bag as an aggregation without an explicit restriction, a sequence is qualified by the *ordered* attribute while the alternative is identified with the *only one* attribute of the relationship.

In the previous figure we have a small portion of a robotic vision ontology. *MonoImage* is a specialization of the *ImData* concept with a time stamp (grabbing time). The ordered aggregation of two mono images gives the *StereoImage*. We define the *GiveImage* action in order to allow a robot to ask for an image. The image should be provided by the *Actor* and sent to the *ResultReceiver* (both agents). Predicates are also defined in relation to some existing concepts (*IsImage*, *IsStImage*).

Communication Ontology Description

The COD diagram is a class diagram and it is mainly composed of two elements: agents and communications.

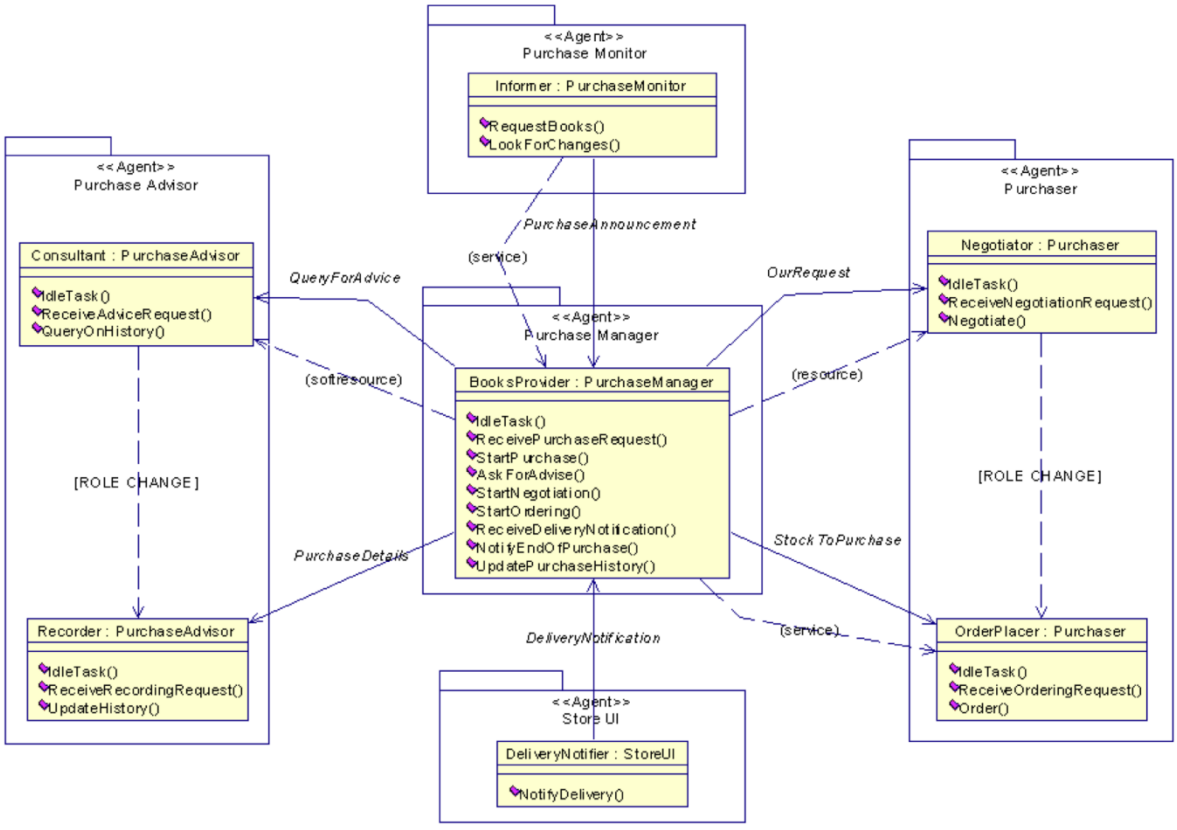


Each agent (fill colour: yellow) is described in terms of its knowledge (pieces of the ontology described in the previous diagram). There is one relationship between two agents for each communication they are involved in. In each relationship the roles played by the agents during the communication are also reported.
Each communication (fill colour: white) is represented by the relationship between the two agents and it is detailed in the relationship attribute class. The class is identified by a unique name (also reported in the relationship among the two agents) and it is described by the *ontology*, *language* and *protocol* fields.
The *ontology* field refers to an element of the DOD (Domain Ontology Description); the *language* addresses for the content language of the communication while the *protocol* points out the adopted FIPA Interaction Protocol.

Agents' definitions are refined by the introduction of the knowledge they need to participate to communications.

In the previously reported picture, the PurchaseManager agent starts a conversation (see QueryForAdvice association class) with the PurchaseAdvisor agent. The Conversation contains the Course ontology, the Query protocol and the RDF language. This means that the PurchaseManager wants to perform a speech act based on the FIPA's query protocol in order to ask the PurchaseAdvisor advice on how to purchase (supplier, number of stocks, number of items per each, purchase-money) provided the Course information.

Roles Description



We represent the Role Description diagram as a class diagram where roles are classes grouped in packages representing the agents.
Roles can be connected by relationships representing changes of role, dependencies for a service or the availability of a resource and communications. Each role is obtained composing several tasks for this reason we specify the tasks involved in the role using the operation compartment of each class.
More in details:
1 Classes represent roles of the agent. They are grouped in packages that stand for the agent.

2 Relationships among roles can be of 3 different kinds:

2.1 Communications. Represented by a solid line directed from the initiator to the participant. Names of communications come from the Communication Ontology Description diagram.

2.2 Dependencies. Like in i*, we can have service or resource dependencies. A *service* dependency shows that a role depends on another to bring about a goal (indicated by a dashed line with the *service* stereotype). In the *resource* dependency, a role depends on another for the availability of an entity (indicated by a dashed line with the *resource* stereotype). We can also have *soft-service* and *soft-resource* dependencies; in this case the requested service/resource is helpful or desirable, but not essential to achieve a role's goal.

2.3 Role changes. This connection is depicted as a dependency relationship because we want to signify the dependency of the second role on the first. Sometimes the trigger condition is not explicitly generated by the first role but its precedent appearance in the scenario justifies the consideration that it is necessary to prepare the situation that allows the second role to start. We use OCL or semi-formal text to express the trigger condition.

The Agent Society Phase

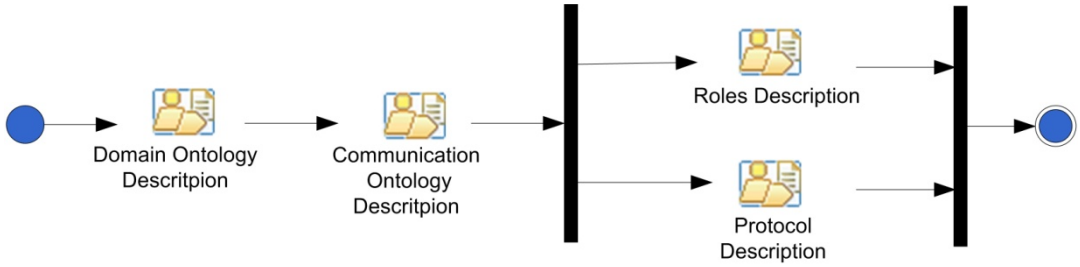


Figure 2 The Agent Implementation Phase flow of activities

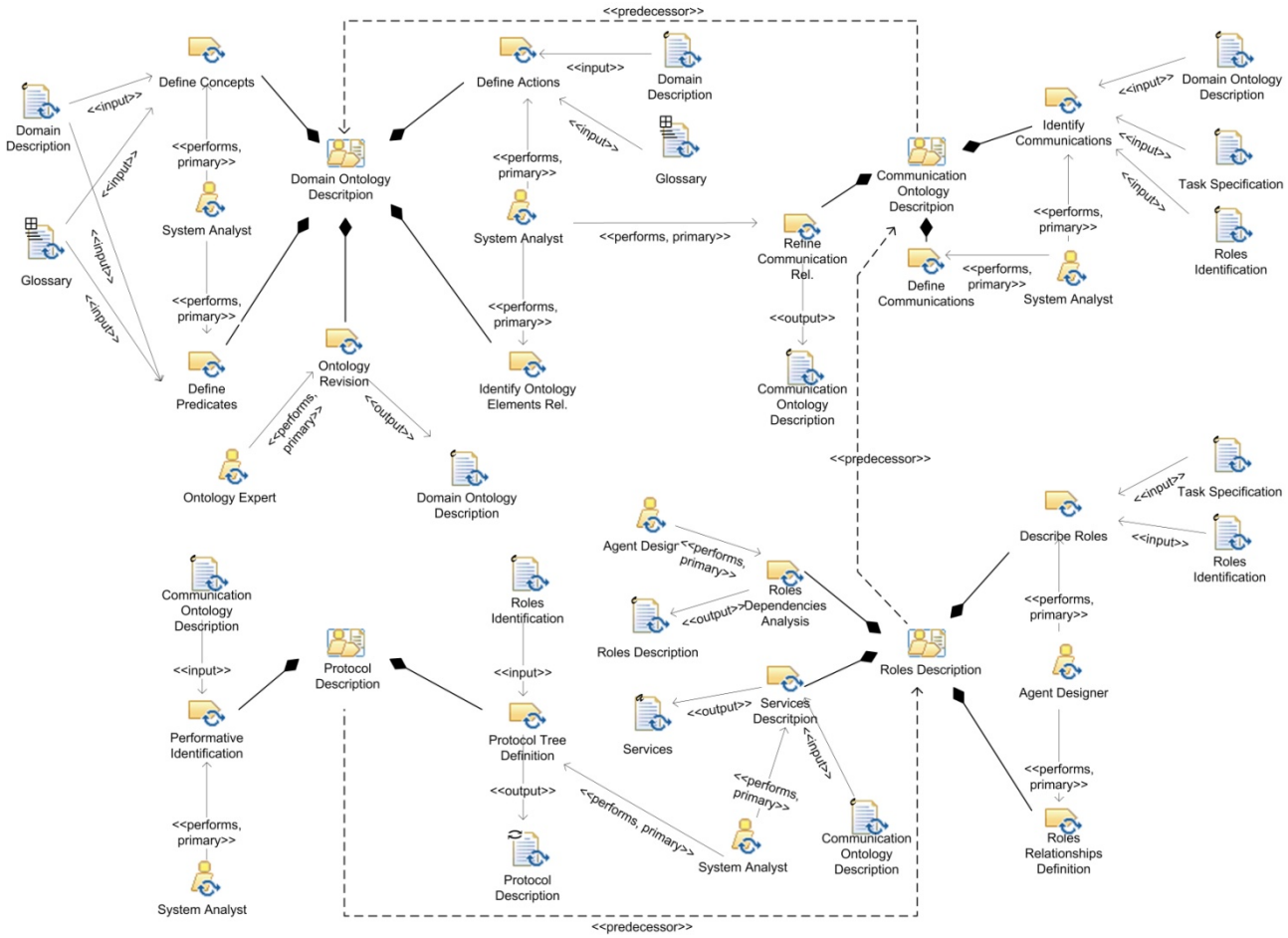


Figure 3. The Agent Implementation phase described in terms of activities and work products

Process roles

System Analyst

Agent Designer

Activity Details

Domain Ontology Description

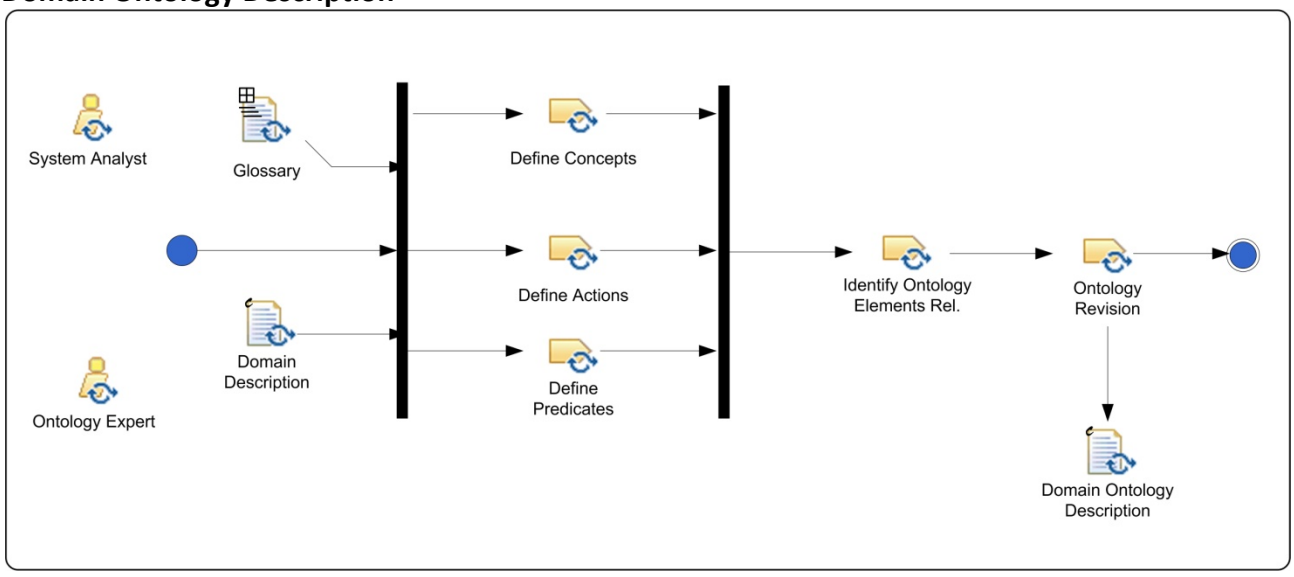


Figure 4. : The flow of tasks of the Domain Ontology Description Activity

Communication Ontology Description

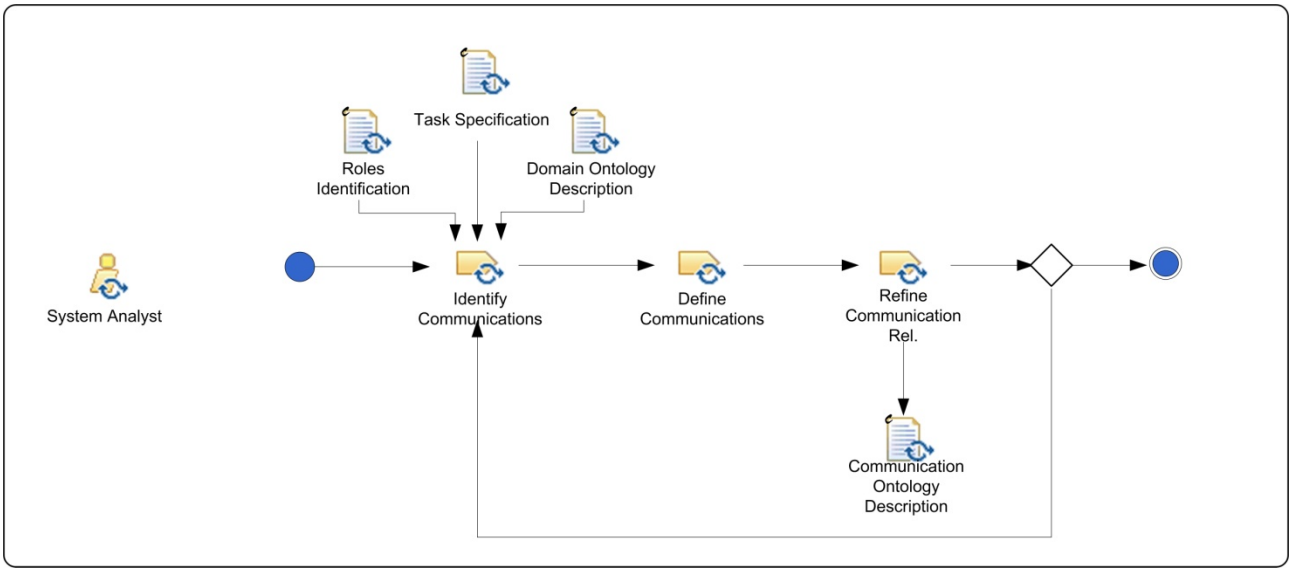


Figure 5. : The flow of tasks of the Communication Ontology Descritpion Activity

Roles Descritpion

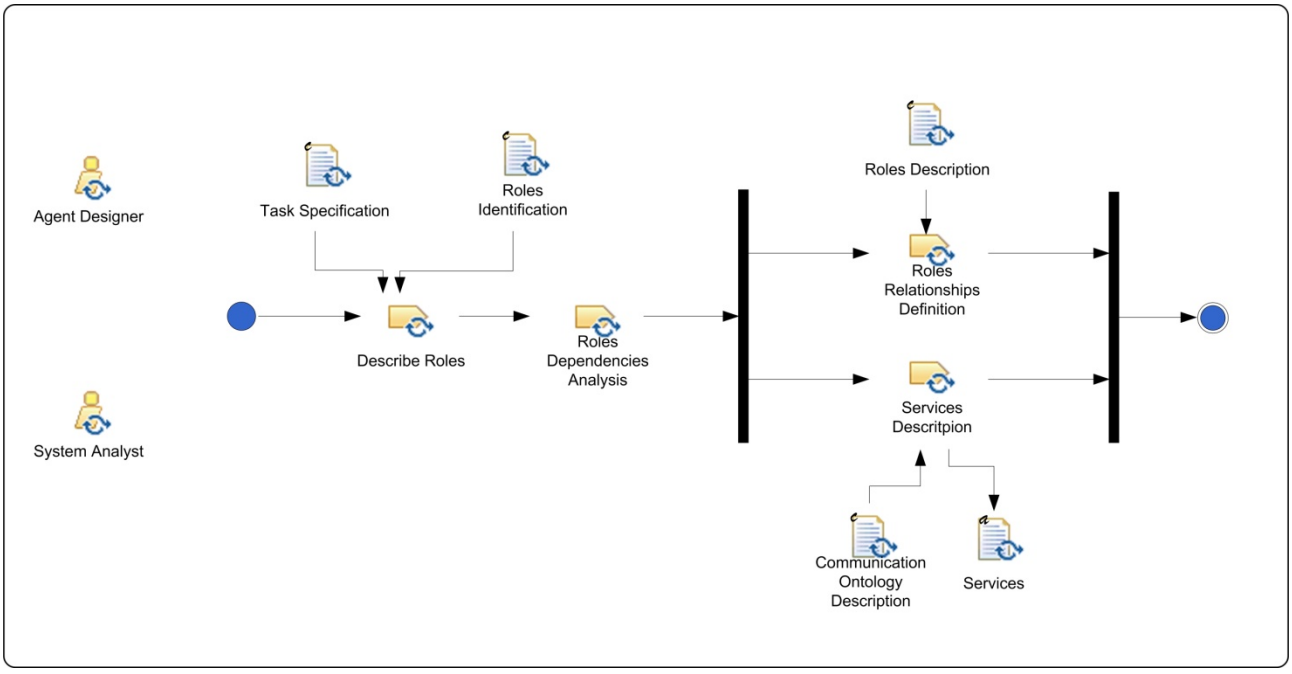


Figure 6. : The flow of tasks of the Roles Description Activity

Protocols Description

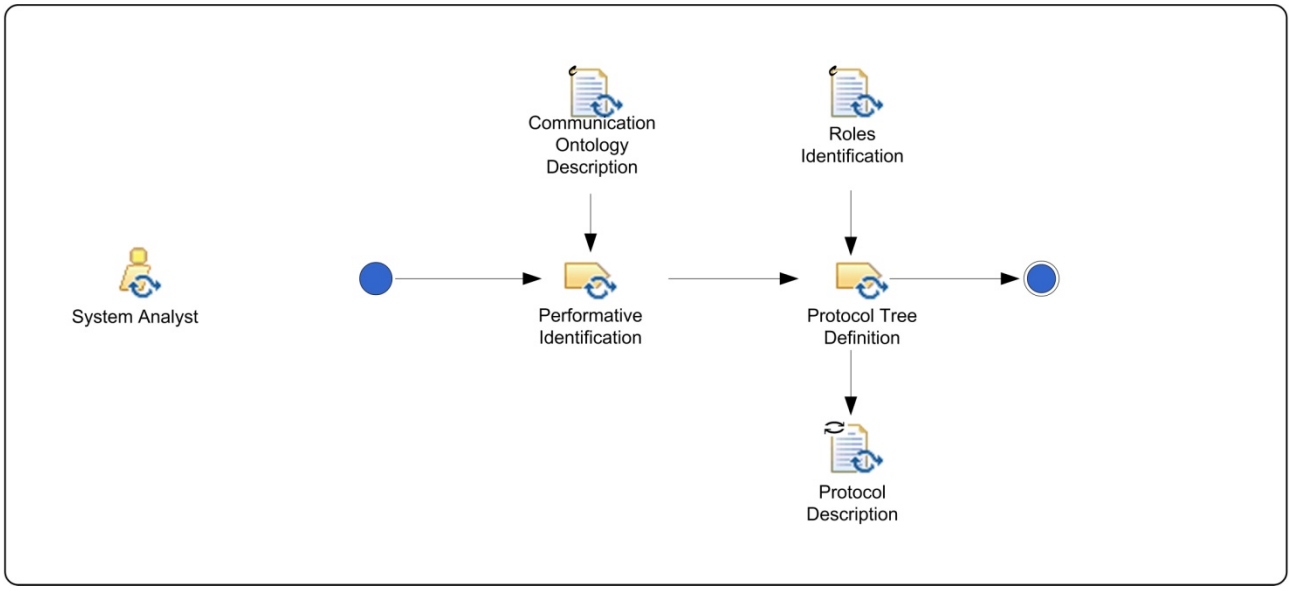


Figure 7. ***: The flow of tasks of the Protocols Descritpion Activity***

Work Products

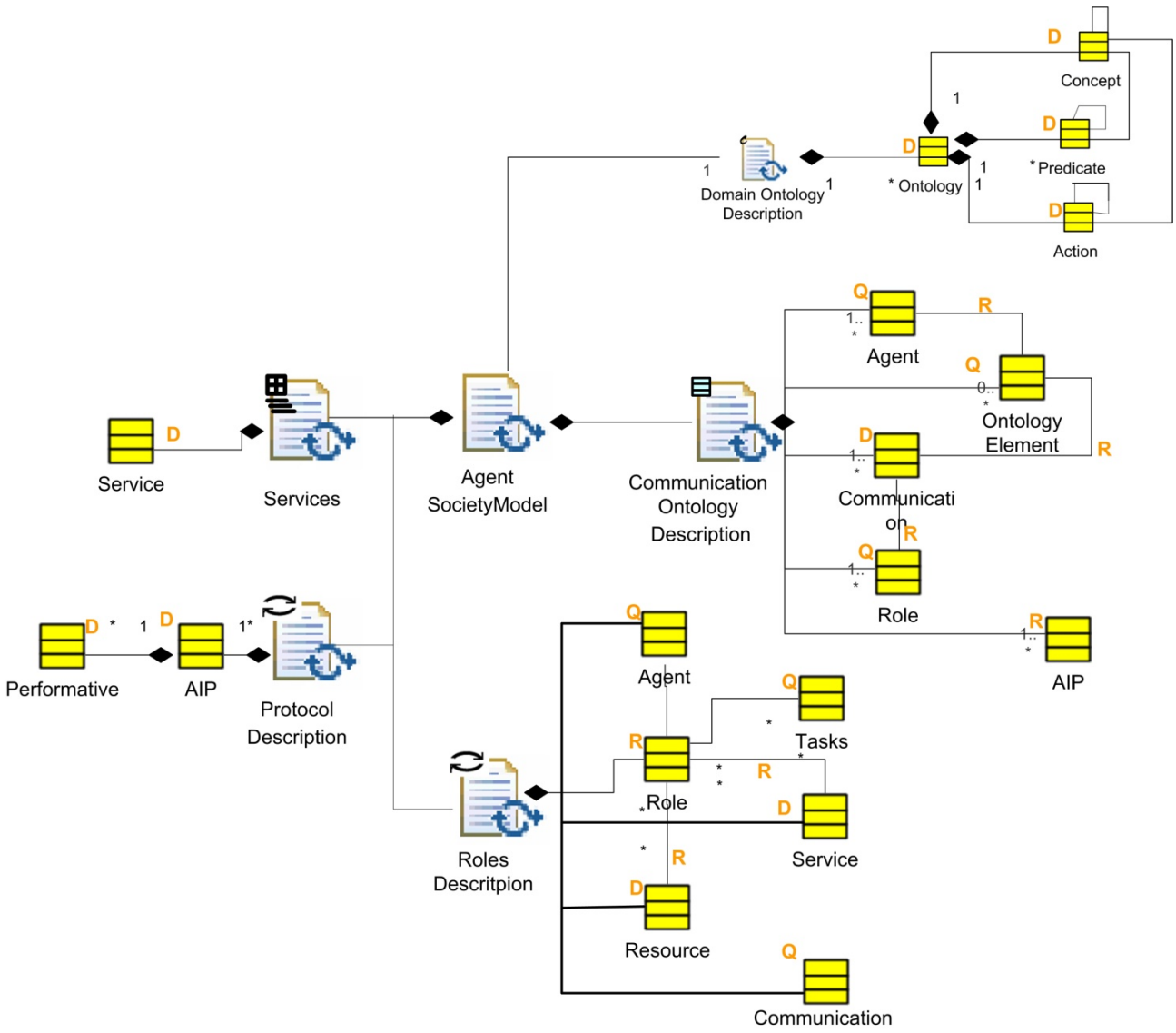


Figure 8. The Agent Society Phase documents structure

Work Product Kinds

The Code Phase

The Deployment Phase

Work Product Dependencies

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