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

PASSI2 - Going Towards Maturity of the PASSI Process

Massimo Cossentino, Valeria Seidita

RT-ICAR-PA-09-02	December 2009

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

PASSI2 - Going Towards Maturity of the PASSI Process

Massimo Cossentino[1], Valeria Seidita[2]

Rapporto Tecnico N.:	*Data:*
RT-ICAR-PA-09-02	**December 2009**

[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).

Index

Introduction

The PASSI2 design process is the evolution of PASSI (Process for Agent Society Specification and Implementation) [44], it covers all the phases from the requirements analysis to the deployment configuration, coding, and testing.

PASSI2, so as PASSI and all its evolution, is based on a meta-model describing the elements that constitute the system to be designed (agents, tasks, communications, roles) and what are the relationships among them. The importance of this description is in the lack of an universally accepted meta-model of MASs (differently from object oriented systems) that makes unclear any agent design process that does not precisely define the structure of the system it aims to produce.

PASSI2 has been designed keeping in mind the possibility of designing systems with the following peculiarities: (i) highly distributed, (ii) subject to a (relatively) low rate of requirements changes, (iii) openness (external systems and agents that are unknown at design time will interact with the system to be built at runtime). Robotics, workflow management, and information systems are the specific application areas where it has been applied.

As regards the implementation architecture, since we consider remarkably important the adoption (and enhancement) of standards for the diffusion of the agent-base software engineering, we decided to use the FIPA architecture although; this however, does not mean that PASSI cannot be largely applied for the design of non FIPA agents (like BDI ones).

Actors involved in the design process are supposed to be designers with:

- some experience of object-oriented design (using processes like the Unified Process [RUP,UP]). Starting from this requisite we propose a process that relies on common concepts like a functionality-oriented requirement analysis (differently from methodologies like Tropos [26] that is goal-based). This allows a smooth change towards AO approaches for a great number of already skilled designers that could profitably reuse their past experience without the significant slowdown induced by a remarkable change in their mental attitudes.
- a good knowledge of UML and the use of related CASE tools. All the diagrams used in PASSI2 are based on the Unified Modeling Language and their syntax has been modified only to satisfy the needs of agents representation in a few limited cases. Sometimes the significance of their elements is mapped to agent-related concepts using stereotypes.
- some kind of confidence with agent-oriented solutions. PASSI2 supports different levels of details and therefore it is recommended that the designer has some knowledge about the implementation of agent systems (i.e. implementation frameworks, languages and related basic concepts). In our experiences, the difficulties usually reported by OO designers are mainly related with agent design aspects like ontology, communications, and behavior architecture.

Another specific feature of PASSI2 is that it is one of the few methodologies that incorporate the design of ontology. This because we think that ontology design is a fundamental step in MAS design and the designer's vision of the domain expressed in terms of an ontology influences all of his work.

A Quick Overview on the PASSI2 Process

In the following sections, we will introduce the elements of the PASSI2 design process. It is composed of three models (Figure 1) that address different design concerns and nineteen phases in the process of building a model.

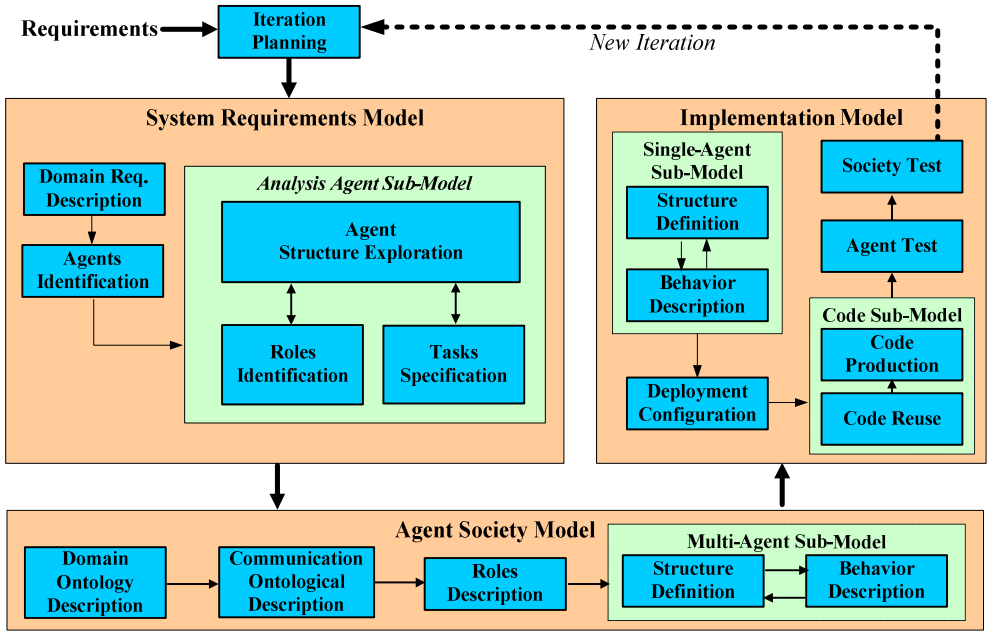

Figure 1 . The models and phases of the PASSI2 methodology

In PASSI2 we use UML and adapts it to the need of representing agent systems through its extension mechanisms (constraints, tagged values and stereotypes). Within the agent design community, a variant or dialect of UML, (AUML [15]) has been proposed as a standard but it is not sufficient for our aims because of its by now limited scope (it re-defines the semantics and syntax of only a few different diagrams, not the whole UML set of views). Only very recently, a more extensive language (AML [43]) has been presented and in the next future we will evaluate it for a possible adoption in the PASSI2 artifacts. This however will not have a drastic impact on the PASSI2 process since the process itself is well separated by the syntax adopted for producing the prescribed artifacts.

The models and phases of PASSI2 are:

1. **System Requirements Model**. A model of the system requirements in terms of agency and purpose. Developing this model involves five steps:
   - Domain Description (DD): A functional description of the system using conventional use-case diagrams.
   - Agent Identification (AId): Separation of responsibility concerns into agents, represented as UML packages.
   - Role Identification (RId): Use of sequence diagrams to represent each agent's responsibilities through role-specific scenarios.
   - Agent Structure Exploration (ASE): An analysis-level description of the agent structure in terms of tasks required for accomplishing the agent's functionalities.
   - Task Specification (TSp): Specification through state/activity diagrams of the capabilities of each agent.
2. **Agent Society Model**. A model of the social interactions and dependencies among the agents involved in the solution. Developing this model involves three steps:

- Domain Ontology Description (DOD). Use of class diagrams to describe domain categories (concepts), actions that could affect their state and propositions about values of categories.

- Communication Ontology Description (COD). Use of class diagrams to describe agents' communications in terms of referred ontology, interaction protocol and message content language.
- Role Description (RD). Use of class diagrams to show distinct roles played by agents, the tasks involved that the roles involve, communication capabilities and inter-agent dependencies in terms of services.
- Multi-Agent Structure Definition (MASD). Use of conventional class diagrams to describe the structure of solution agent classes at the social level of abstraction.
- Multi-Agent Behavior Description (MABD). Use of activity diagrams or state-charts to describe the behavior of individual agents at the social level of abstraction.

3. **Implementation Model**. A model of the solution architecture in terms of classes, methods, deployment configuration, code and testing directives; it is composed of ten phases, the first two are performed at both the multi-agent (whole agent society) and single-agent abstraction level:

- Single-Agent Structure Definition (SASD). Use of conventional class diagrams to describe the structure of solution agent classes at the implementation level of abstraction.
- Single-Agent Behavior Description (SABD). Use of activity diagrams or state-charts to describe the behavior of individual agents at the implementation level of abstraction.
- Deployment Configuration (D.C.). Use of deployment diagrams to describe the allocation of agents to the available processing units and any constraints on migration, mobility and configuration of hosts and agent-running platforms.
- Code Reuse (C.R.). A library of patterns with associated reusable code allows the automatic generation of significant portions of code.
- Code Completion (C.C.). Source code of the target system is manually completed.
- Agent Test: it is devoted to verifying the single behavior with regards to the original requirements of the system solved by the specific agent.
- Society Test: the validation of the correct interaction of the agents is performed, in order to verify that they actually concur in solving problems that need cooperation. This test is done in the most real situation that can be simulated in the development environment.

The **Iteration Planning** activity is someway positioned at an higher level of abstraction, above this logical sequence of models and phases. It is at the base of every iterative incremental process and in our case consists in the analysis of the *Problem Statement* and all the other available documents (for instance outputs of previous iterations) in order to identify the requirements (and related risks) that should be faced in the next iteration.

The PASSI Process Evolution

During these years of work and experiences using PASSI, we gradually tuned it in order to best fit the designer needs. The most important changes we did are:

1) Ontology description split into Domain Ontology Description + Communication Ontology Description

2) Elimination of Protocol Description

3) Test introduction

4) Deployment configuration/System integration different meanings

The PASSI2 Process

In this section we will discuss the steps of PASSI2, referring to the well known standard FIPA architecture [10], [11] as a deploying environment. We will describe the different activities accomplished by the designer and the consequent artifacts using an agentified implementation of the same PASSI2 process as a case study. The idea that is behind this approach is describing PASSI2 with PASSI2.

System Requirements Model

Its aim is to produce an analysis level description of the system in terms of requirements and agents. The corresponding discipline agent is responsible for maintaining the analysis model that will be an input for the following phases. It includes the five phases that will now be discussed more in details.

In Figure 2 we can see a description of the System Requirements model activities and related artifacts.

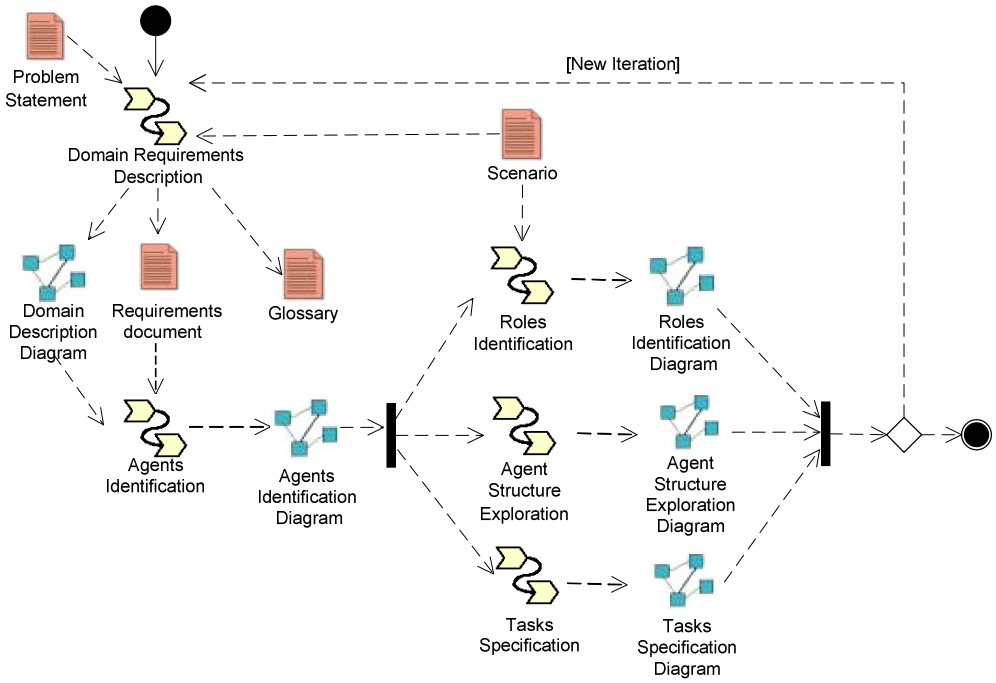

Figure 2. The System Requirements Model activities and resulting artifacts

Starting from the Problem Statement document, the designer during the Domain Requirements Description phase produces a decomposition of the system requirements in terms of use cases. These are later grouped into packages that represent the responsibilities of each agent in terms of what functionalities it has to realise. At this stage agents are identified from a functional point of view and an initial analysis model can be draft from both the structural (Agent Structure Exploration phase) and behavioural (Roles Identification and Task Specification phases).

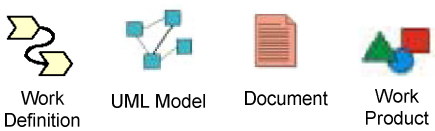

Figure 3. UML profile for SPEM symbols (as specified in [27])

According to the UML profile specified for SPEM in [27], in this diagram (and in some of the following ones) we will use the symbols reported in Figure 4 with the same meaning prescribed by SPEM specifications [27]: work definition is a kind of operation that describes the work performed in the process, UML model and document are kinds of work products.

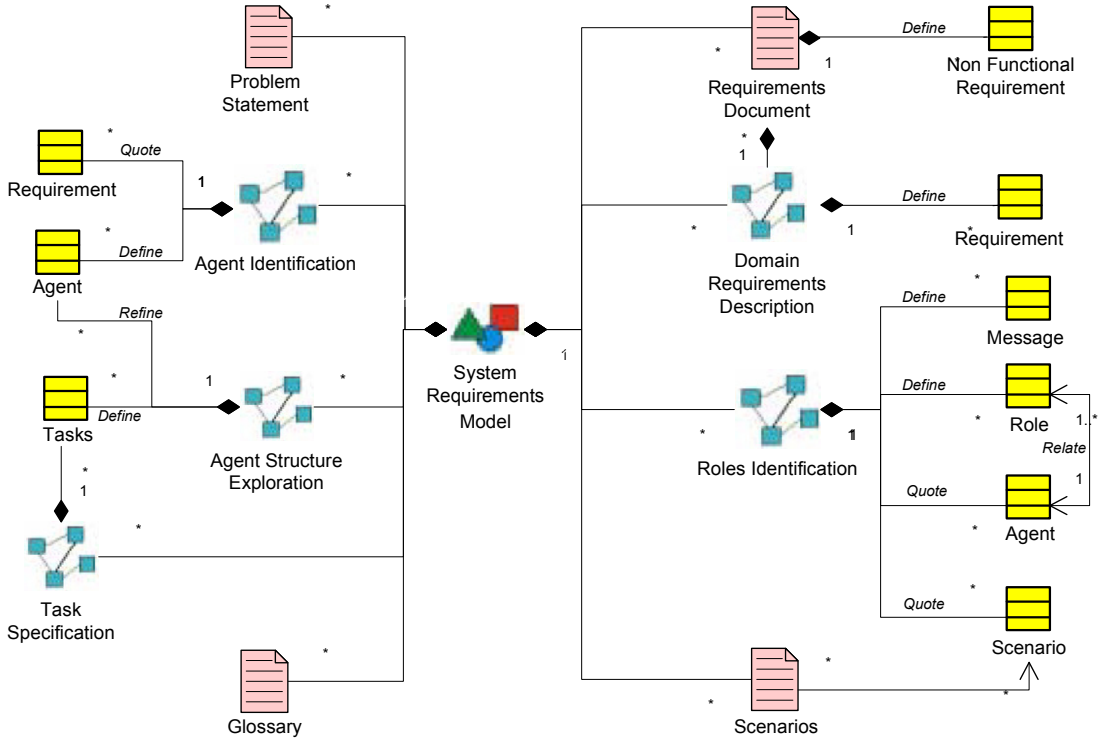

Figure 4. The structure of the artifacts for the System Requirements model

The mapping between the elements of the MAS meta-model described in sub-section and the artefacts of this discipline is described in Figure 4. This diagram represents the System Requirements model in terms of UML models and text documents. Each of them reports one or more elements from the PASSI2 MAS meta-model; each MAS meta-model element is represented using an UML class icon (yellow filled) and in each documents such elements can be *defined* (this means that the elements are introduced for the first time in the design in this artefact), *refined* (this means that an already defined element is detailed or someway updated) or simply *quoted* (this means that the elements have been already defined and are reported in this artefact only to complete its structure but no work has to be done on them). Obviously because of the iterative nature of the PASSI2 process, if an element is to be defined in a specific phase, it is possible that in one of the following iterations it will be changed (refined) according to the evolution of the design.

In the next subsections we will detail the different phases of this model.

Domain Requirements Description Phase

Requirements elicitation may involve the elaboration of system goals [17][18][19][20], and this can be seen as even more appropriate in MAS design (as proposed in Tropos [26]), which are teleological in nature. DeLoach et al. [1] describe requirements using goals, which they refine into a system description using use-case and sequence diagrams.

Despite several authors have a commitment to the use of goals in requirements engineering, we have chosen a different approach. In PASSI2, we describe requirements directly in terms of

use-case diagrams that are obtained either through standard requirements elicitation precursors to object-oriented (OO) methods [6][7] because we think that this approach allows an easier transition to designers coming from the OO world (students of our basic software engineering courses among them).

In this phase we describe the system requirements in terms of use cases in order to prepare the Agent Identification phase where these elements will be clustered and put under the responsibility of some agent. If a goal-oriented approach is used at the beginning of the design (like for instance the early requirements analysis proscribed by Tropos [26]), requirements should be derived from it and then applied to this phase.

As a result, a functional description of the system is provided through a hierarchical series of use-case (UC) diagrams. Scenarios that are at the basis of the UC diagrams come from the Problem Statement document and they are explained using sequence diagrams in following phases.

The expected output of this phase is an UML use case diagram and a textual document 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.

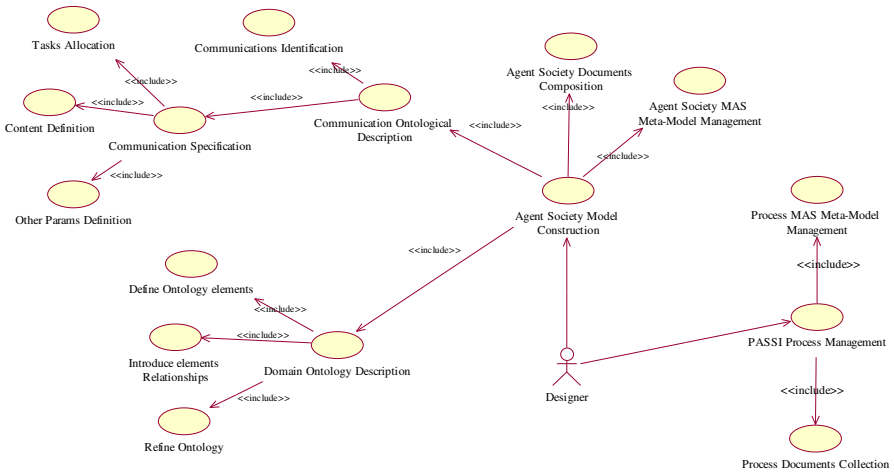

Figure 5. The Domain Requirements Description (DRD) diagram

In Figure 5, an example of DRD diagram describes (some of) the activities of the PASSI2 design process (these could be thought as the description of the functionalities of a tool that supports the development process with PASSI2). For instance, consider the *Communication Ontological Description* phase where the designer identifies and describes each communication that takes place among agents. In Figure 9 we can see that this phase is part of the Agent Society model; the fundamental use case of this model is *Agent Society Model Construction* that also includes the functionalities related to other phases like the *Domain Ontology Description*, and some other model-level features like the *Agent Society Documents Composition* use case and the system model management (*Agent Society MAS Meta-Model Management* use case). Finally, all the models (disciplines) are in turn coordinated by a process level use case (*PASSI2 Process Management*)

It is worth to highlight that use cases identification, their refinement and relationships introduction is not considered within the scope of the PASSI2 methodology since we suppose this is part of the designer background skills and any of the existing approaches can be considered valid if it produces a good representation of the system requirements in this form.

Agents Identification Phase

One of the central issues in PASSI2 is the identification of agents early in the development process. If agents were merely MAS implementation components, their identification during the analysis of requirements would be premature. However, if a MAS is a society of anthropomorphic entities, it is more reasonable to divide the description of required behaviors into loci of responsibility from the start. This is particularly true in cases where the "system" is an heterogeneous society of intended and existent agents that in Jackson's terminology can be "bidden" or influenced but not deterministically controlled [21].

In this way, the designer skill in capturing system requirements has been capitalized in order to produce an initial representation of the system functionalities and now this model is used to identify agents and designate their responsibilities in terms of requirements to satisfy.

Agent identification starts from use-case diagrams of the previous step. A direct correspondence is created among the requirements and the agents that will be created. Each use case is assigned to an agent (use cases assigned to agents are clustered in one package labeled with the same agent name) whose mission becomes satisfying the corresponding requirements by its own or asking for collaboration from the others. The need for this collaboration, can be the obvious consequence of assigning two use cases related for instance with an include relationship to two different agents. The first agent (that owns the base use case) will ask to the other (that owns the included use case) to provide the functionality it is lacking of. This establishes the bases for creating a society of collaborating agents whose common goal is the success of the whole social system through the satisfaction (when this is not in contrast with the overall goal) of the individual objectives.

The resulting artifact is a use case diagram (or a set of such diagrams) reporting the same use cases of the previous phase now clustered in a set of packages, each one representing one agent. As it is common, we represent external entities interacting with our system (people, not agent-based software systems and external agents) 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's 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 among 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 is to be changed to *communication* since a conversation is the unique interaction way for agents (see the relationship among use cases *Agent Society Model Construction* and *Communication Ontological Description* in Figure 5 and Figure 6). This is a necessary extension of the UML specifications that allow communication relationships only among use cases and actors. The direction of 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 (this could happen, for instance, when an agent responsible for some kind of web search activity instantiates a lot of spiders assigning a different site to each one); in this case, that is however not frequent, we use an *instantiate* stereotype to distinguish this situation from the

others (obviously it in not always possible to identify this difference in the agents' relationship at this early stage of the design).

- An agent's knowledge can increase through communication with other agents or exploration of the real world (sensing activity).

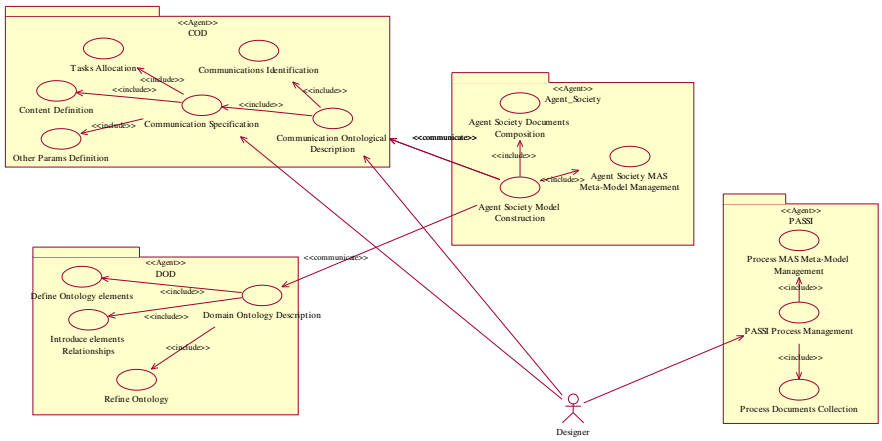

Figure 6. An example of Agent Identification diagram

Communications among agents may differ significantly from those between agents and other entities. Among agents, communication patterns follow a specific protocol and communication content refers to a specific ontology. Communications between external actors and agents are not necessarily implemented like that. For example, agents could receive information directly from actors through sensing devices, whose communication protocols are constrained by the hardware devices and their drivers.

No precise rules exist to perform the activities related to this phase but using our experience with it we can draw some guidelines. The first activity consists in analyzing the use case diagrams resulting from the previous phase and attempt their clustering in a set of packages according to these criteria:

- 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 is a little difficult at this stage since data flow is not evident in use case diagrams)
- 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 not be accessed by many other parts of the system and should be isolated by the remaining part of the system eventually using a wrapping solution.

Roles Identification Phase

During this phase we are concerned with an agent's externally visible behavior, so any representation of its internal behavior is approximate. In PASSI2 a role is defined by the set of responsibilities defining the subjective behavior of an agent in an interaction (conversation) with another or in providing some service in one or more scenarios; an agent may play one or more roles at the same time.

Because of the interaction that is behind any conversation or service provided, the role can be regarded as a social manifestation of the agent behavior.

The Roles Identification phase produces a set of sequence diagrams that specify scenarios from the agents' identification use case diagram. In this context, it is particularly important to investigate all the paths involving inter-agent communications, and fortunately some guidelines can be considered: (1) such communication paths are shown in the A.Id. diagram by the presence of a relationship between two agents with the communication/instantiation stereotype; (2) each relationship may belong to several scenarios; (3) for each relationship in a specific scenario of the A.Id. diagram, there is at least one message in the sequence diagram of the R.Id. phase.

In this phase, roles are identified in the sense that agents' external manifestations are captured in sequence diagrams where agents participate playing one or more roles concurring to the evolution of the system dynamic.

It is worth to remind that in object-oriented approaches, many designers use to introduce boundary, control and entity objects in order to classify elements populating scenarios according to their responsibility. Interface objects are responsible for interactions with external actors, control objects supervise the flow of control within the scenario, and entity objects are used as data stores.

These categories lose a great part of their meaning when describing agents. In fact, it makes no sense to have a controlling element in a scenario when all the participating agents are naturally autonomous and the correct execution of the scenario is conditioned by the concurrent will of all the participants (the most similar element could be some kind of coordinating role). Entity objects have not a direct counterpart in MASs since each agent is, generally speaking, a complex element of design with a consistent part of knowledge by its own. The situation of interface objects is different because they could be personified by GUI agents that are often used to implement with a light (and easily movable) agent the graphical interface between the user and a complex heavy agent that is not easily deployable in some kind of devices. In this sense, there is an important difference that should be reminded between agents and objects: most agent definitions**Errore. L'origine riferimento non è stata trovata.** add a sensing capability to the agent (the way the agent uses to explore its world) and this can be considered like an interface role too.

As regards the contribution of this phase to the MAS meta-model instantiation, we can see that the following elements are defined:

1. scenarios (described using sequence diagrams),
2. roles (described in terms of name, interactions with other agents/roles and responsibilities),
3. which roles participate in each scenario.

The different roles that an agent can play are introduced as objects in the appropriate sequence diagram. An agent may participate in several scenarios playing distinct roles in each, and may even appear more than once in a single sequence diagram playing different roles.

Differently from object-oriented design where messages represent methods invocation, messages reported in these diagrams have no relationship with agent's internal structure since we are here representing an external view of the agents behavior and we have no information about the way each agent will deal with its ongoing communications (and composing messages); obviously a task will be related to one or more messages in order to process them, but this level of description is out of the scope of this phase just like it is in the AUML sequence diagram [15].

It is interesting to note that contrarily of what happens with object systems, during the design of agent scenarios, it is possible to have a sequence diagram without any actor. This is the logical consequence of the agent autonomy; this means that an agent could trigger a scenario without the participation of any external actor.

Task Specification Phase

In the previous phases we identified agents and their roles. Now, in the task specification phase we are facing the third element of our MAS meta-model: tasks.

While use case diagrams in the Agent Identification phase represent a functional description of the multi-agent collaboration (the whole agent society), in the Task Specification phase we look at one agent at a time, drawing one activity diagram for each agent in order to represent the activities performed by the agent to carry on its duty. Such a diagram represents the plan of the agent, reporting the relationships among the external stimuli received by the agent and its behaviour (expressed in terms of fired tasks).

Because of the specific nature of an activity diagram (a kind of state chart), it is very easy to model aspects like concurrency (several behaviors are executed concurrently in order to realize some intention), control structures (agent's decisions) and incoming events (messages from other agents but also external events) and agent's states (for instance "waiting for transaction commit" can be represented as object states [36]). Contrarily, it is not easy to represent in this way other situations like non deterministic flow of control that could arise in some agents implementation.

In some cases a state chart diagram should be preferred to the activity one; this is the case of agents based on the Brook's Subsumption architecture [34][35] where the overall system behavior is the result of many interacting simple behaviors organized in layers. Each layer provides a set of out-of-the-shelf behaviors and higher levels compose them to perform complex behaviors. Layers of the Subsumption architecture are modeled as networks of finite state machines augmented with timers (Augmented Finite State Machines, AFSMs), and therefore, they can be naturally modeled with UML state diagrams. Using this kind of diagrams is not a problem in PASSI2 and it could be done when necessary but in this work we will only describe the use of activity diagrams since we used them more frequently.

Another common agent architecture that should be considered is the DBI (Belief, Desire, Intention) one [37]. Several attempts have been done to implement the BDI agent architecture ([32][33]) with FIPA agents ([28][28][30][31]) and we think that an activity diagram could help in preparing an initial hypothesis of what the BDI agent plan can be.

As proposed in the classical Procedural Reasoning System (PRS) [37], agent's desires and intentions are realized by using plans from a library. We suppose that the responsibility of realizing those plans is delegated to the agent's tasks and a first hypothesis of the control flow that is behind their execution is presented in the TSp diagrams; the strategy will later be consolidated within the following phases and the portion of plan related to each task will be detailed in the implementation model of the task itself.

Other BDI elements can be modeled in the following phases, for instance, belief can be seen as the instance of knowledge structure (described in the DOD phase) hold by an agent, desires can be deducted from the set of agent's functional (and eventually non functional) requirements since each agent wants to accomplish its duty (we are dealing with rational agents [37] who will naturally act in accordance to their goals), while intentions can be considered to be the rationale that is behind the T.Sp. diagram.

As regards the Tasks Specification diagrams syntax (in form of activity diagrams), we draw one different diagram for each agent (see Figure 7) where each activity represents one task that the agent can do. Each diagram is divided in two different sections using swim-lanes: the right one reports the tasks of this agent (one activity for each task), the left swim-lane reports the tasks of the agents that interact with this one.

Relationships between the activities represent within-agent communications if the activities are located in the same swim-lane. These are not speech acts; they are often signals addressing the necessity of beginning an elaboration that is left to a specific task (i.e. signals to delegate another task to do something). If a relationship connects elements of two different swim-lanes, it represents a communication from an agent to another one (generally speaking this could be composed of more than one message, at this stage this has not been yet formalized); it is also possible that several related relationships will be later (in the COD phase, see sub-section 0) organized into one single communication. The inner behavior of each task can be specified more in detail by using sequence diagrams, other activity diagrams or semi-formal text.

In Figure 7 we can see the TSp diagram for the *COD* agent introduced in Figure 6. It describes that the *Agent_Society* agent delegates the *COD* one to compose the diagram with the *Start_Composition* message that is received and acknowledged by the *Compose_Diagram* task; according to its plan, the *COD* agent asks for the list of interactions and then compiles the diagram (*Design_diagram* task). This agent also offers the possibility of introducing communication details (*Detail_Communications* task) and checks its pertinence part of the model (*Check_Model* task).

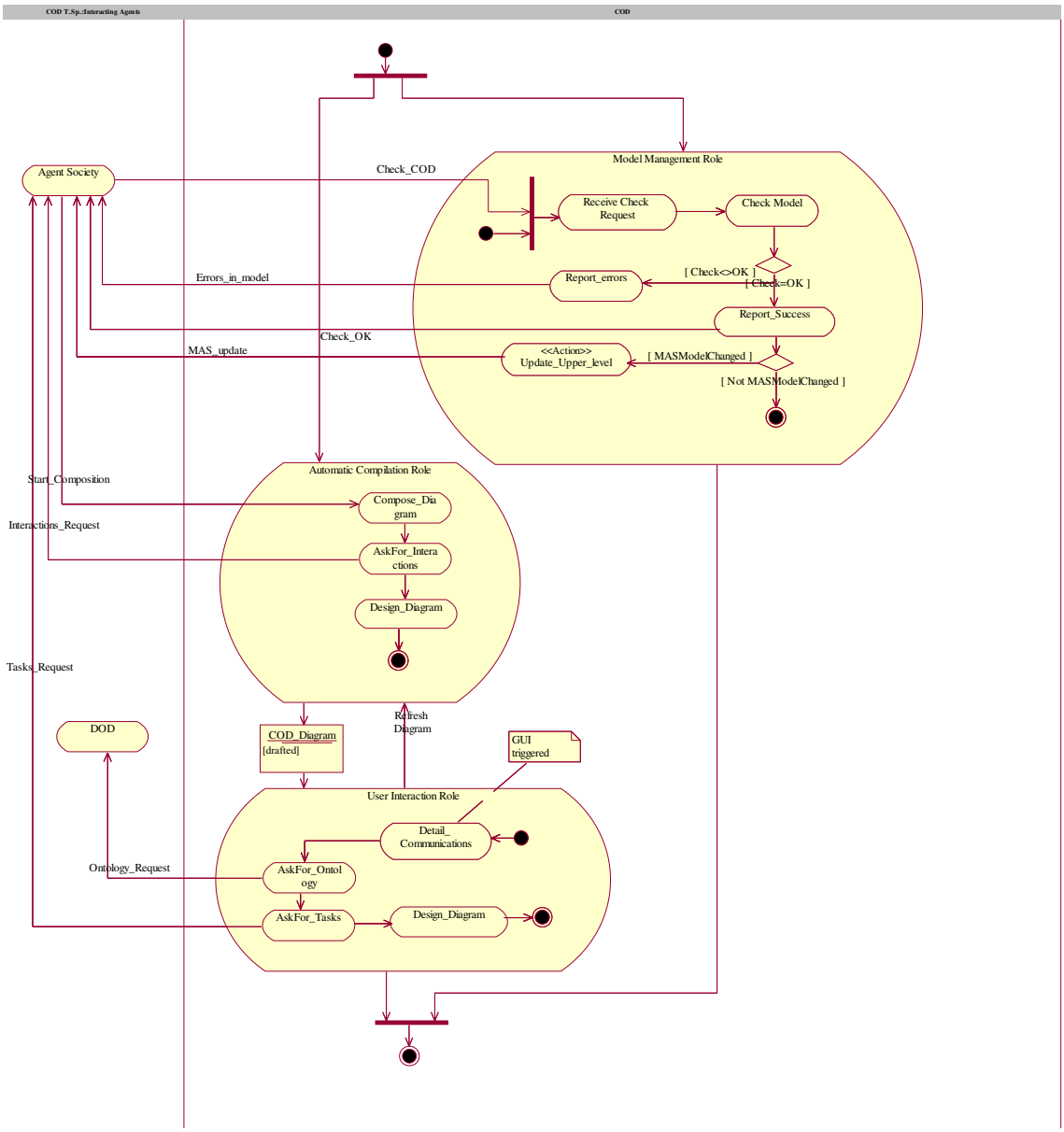

Figure 7. The Task Specification diagram for the COD agent.

In our experiences with the previous version of PASSI2, we found that this is often one of the most difficult phases for not skilled designers; for this reason we suggest the following guidelines: during the initial composition of these diagrams, the designer should introduce one task for each incoming communication deduced from RId sequence diagrams (a communication can be composed of several messages, it is considered incoming with regards to the direction of the first of this messages); similarly, a task is introduced for each outgoing message of the R.Id. sequence diagrams. In subsequent iterations the amount of this tasks and their work load should be carefully evaluated in order to obtain a sufficient degree of optimization. This is not possible at the beginning since such an operation requires a detailed understanding of what is to be exchanged by the agents, which interaction protocol and content language will be used (details defined in the Agent Society model phases).

In addition to the above suggested tasks, the designer may have to introduce other tasks that are not immediately visible in an exterior view of the agent. They could arise spontaneously from a

detailed analysis of the agent's external behavior or from the opportunity of better decomposing the agent's structure (to facilitate reuse, optimize calculations, access external devices).

Agent Structure Exploration Phase

The System Requirements model aims at exploring the problem by identifying an agent-oriented perspective that could enable a straight solution that will be detailed in the following models. This analysis point of view could not be lacking of a structural description of the identified agents that is depicted in form of a class diagram at a social abstraction level (all agents are reported together with their interactions). The Agent Structure Exploration phase, in facts, collects the results of the previous phases and represents the agents in forms of classes and their behavioral capabilities (tasks) as class methods. These tasks are the structural implementation of the external manifestations produced by agents in scenarios of the RId phase. They are both methods used to interact with other agents and to accomplish other agent's duties.

Figure 8 reports a portion of an Agent Structure Exploration (ASE) diagram. Each agent is shown as a class and its tasks are reported as methods of the class. Communications are represented by relationships among agents and their name is used to identify them throughout all the remaining part of the project. Agent knowledge is usually neglected here since at this stage we still lack of any study about its structure and therefore it could be confusing to attempt an hypothesis without the guidance of an ontology exploration of the domain as it will be performed in the following DOD phase (see subsection 0).

This diagram representation associated with the scenarios of RId phase and the agent's behavior plan presented in the TSp phase, compose a complete sketch of the requirements for each agent and enable the following design phases. Agents have been identified in the AId phase and related to requirements, roles and communications have been identified in the RId phase (communications are the logical way of implementing agent interactions found in scenarios), tasks have been sketched iterating among this phase, the RId and the TSp phases.

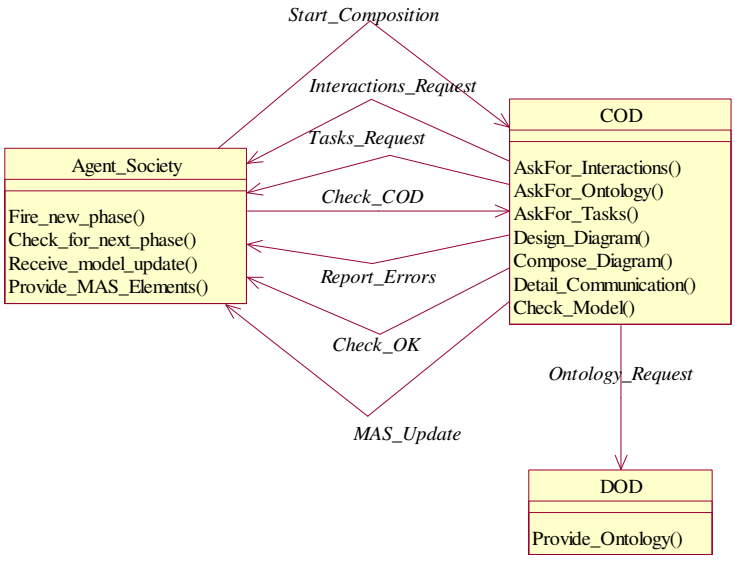

Figure 8. A portion of Agent Structure Exploration Diagram

Agent Society Model

The Agent Society Model introduces an agent-oriented solution for the problem (and related requirements) described in the previous phases. Being agents the center of our approach, the model presents an ontological description of both the domain where agents will live and their communications, then agents are described in terms of roles they play services provided by roles, resource dependencies and finally their structure and behavior.

The activities (and resulting artifacts) that contributes to build up this model are described in Figure 10, where we can identify five phases: Domain Ontology Description (DOD, where the domain of interest is described in terms of its ontology), Communication Ontological Description (COD, where agents communications are described in terms of referred ontology and the other interaction details), Roles Description (where agents are described in terms of their roles and dependencies for services and resource availabilities), Multi-Agent Structure Definition (MASD, where agents are described in terms of the behaviors they own), Multi-Agent Behavior Description (MABD, where agents are described in terms of their behavior both from the social-exterior point of view and the internal flow of control).

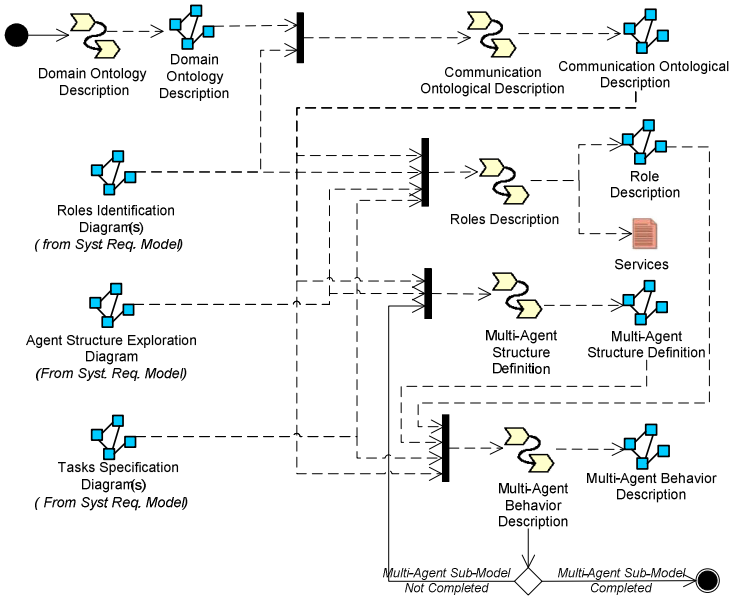

Figure 9. The Agent Society Model activities and resulting artifacts

The relationships among the artifacts produced during these phases and the elements of the PASSI2 MAS meta-model are described in Figure 10 where we can see that one different UML model is created in each of the phases and moreover a text document is written in the Roles Description phase to describe services provided by agents.

In this figure we adopted the same syntax as in Figure 3 but an extension is now necessary: some artifacts relate different elements of the model (for instance this happens in the Communication Ontological Description diagram, where communications are related to ontology elements, content language and AIP); to represent this situation, Figure 10, we introduce the *relate* relationship; it means that one of the activities the designer will perform in producing the artifact consists in relating some instances of the two elements.

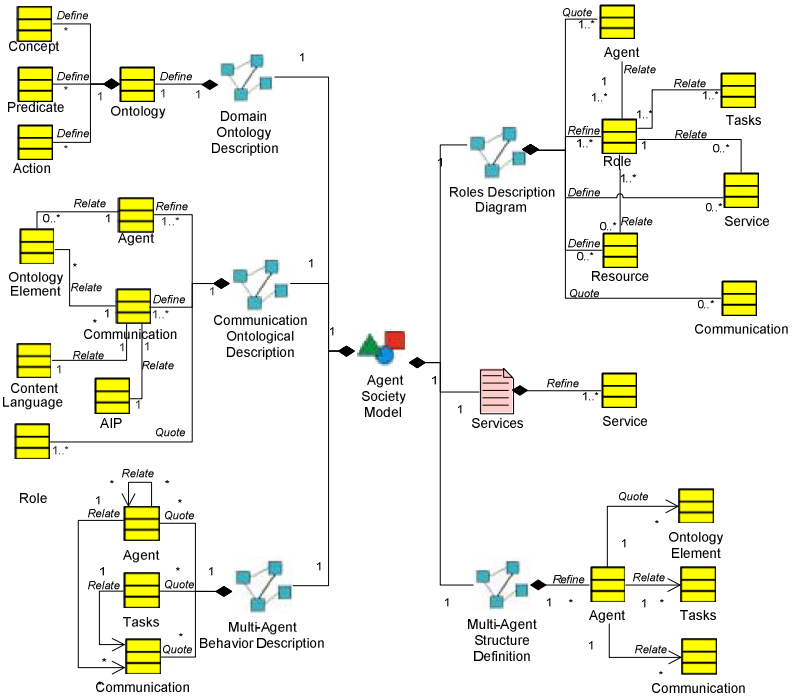

Figure 10. The artifacts structure for the Agent Society Model

Domain Ontology Description Phase

An agent-based system may achieve its semantic richness through explicit ontologies, or domain-specific terminologies and theories. In order to detail the resulting ontology we introduce the DOD (Domain Ontology Description) phase in which we use a class diagram to describe the ontology of the domain representing the involved entities through classes.

PASSI2 is one of the few MAS design processes that include the design of ontology as an explicitly prescribed phase. This is the consequence of a precise choice since in our opinion, a good exploration of the domain is fundamental in order to deeply understand the problem and to have a good adherence between the application domain and the solution to be developed, particularly if the solution will be an agent-based one. For this reason the model we produce of ontology includes concepts (categories, entities of the domain), predicates (assertions on properties of concepts) and actions (that agents can perform in the domain).

Starting from the positions of other authors [12][13], we express ontology using class diagrams. Because of our particular attention for the automatic production of as much code as possible we describe the elements that constitute the domain ontology as an XML schema. In this way we can easily obtain an XML description of the message content expressed in one of the possible ACL (Agent Communication Language) languages [22] (PTK already supports the automatic generation of the ontology code in RDF that is part of both the W3C [38] and FIPA [39] specifications).

In PASSI2, it is important not only modeling the concepts of the domain but also the interactions of the agents with them. With interactions we mean the actions that agents can perform in the environment (using or affecting the constituting elements of the environment itself) and the predicates that can the knowledge of the agent on it.

The basic data model of the RDF ontology (as proposed for standardization in [38]) is based on three object types:

- Resources: things described in the RDF expression;
- Properties: a characteristic, attribute or relation of a resource;

- Statements: it expresses a belief about the resource and it is composed of three parts that are the subject (a resource), the predicate (the property of the subject) and the object (the property value).

This specification has been introduced in FIPA documentation as one of the supported content language (FIPA RDF Content Language [39]). Because of the specific context, some little refinement are to be considered. The W3C RDF resource is named *Object* in FIPA RDF and the W3C RDF statement corresponds to the *Proposition* in the FIPA language.

Multi-agent systems communications are motivated not only by the need of exchanging information but often by the will of the initiator of asking for some kind of collaboration (for example delegating an action). In order to support these conversations, actions have been introduced in the FIPA RDF language. An action is an activity that can be done by an agent. It is described by an act (the operative part of the action), an actor (the entity that is responsible for executing the action) and one or more optional arguments (if necessary for the execution of the action).

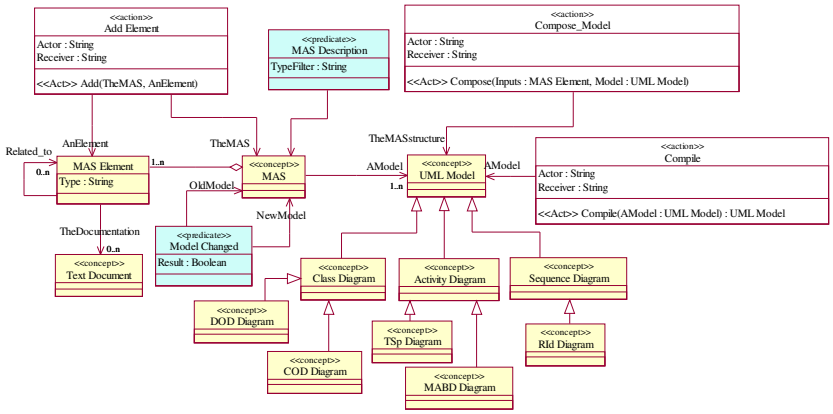

Figure 11. A Domain Ontology Description (DOD) diagram

In Figure 11, a portion of the ontology used to describe the domain related to the PASSI2 toolkit chosen as a case study for this paper, is reported. The diagram (an UML class diagram) represents concepts, predicates and actions with the following conventions: concepts are represented as classes with the *concept* stereotype and with the yellow fill color, predicates and actions are represented by classes with the *predicate* or *action* stereotype and with the blue or white fill color.

Concepts can be related using three UML standard relationships:

- **Generalization**: it permits the generalization/specialization relationship between two concepts that is one of the fundamental operator for constructing an ontology.
- **Association**: it models the existence of some kind of logical relationship between two concepts. It is possible to specify the role of the involved entities in order to clarify the structure.
- **Aggregation**: it can be used to construct sets where value restrictions can be explicitly specified; in the W3C RDF standard three types of container object 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 representing a bag as an aggregation without any explicit restriction, while a sequence is qualified by the *ordered* attribute and the alternative is identified with the *only one* attribute of the relationship.

Exploring Figure 11, we can see that its elements define the pieces of knowledge of the agents and the ontology of their communications. For instance, we can there find the *MAS* concept that represents the MAS model instantiated by the designer during his work; it is composed by MAS elements (like agents, roles, tasks and all the other elements of the MAS meta-model. Now let us consider this interaction occurring in a scenario involving the *COD* agent (*User_interaction* role) and the *DOD* agent (*Ontology_server* role). The first one (*COD* agent) needs a list of the already defined ontology elements in order to allow the definition of the referred ontology attribute of each communication. It sends a request to the *DOD* agent that answers with the *MAS Description* predicate; this latter describes the MAS model by listing its elements as it was requested. During the *Communication Ontological Description* phase, the designer could introduce a new communication between two agents and this corresponds to delegating the *COD* agent of performing the *Add Element* action described in Figure 12; it has two attributes: the actor and the receiver. As already said, this action will be performed by the *COD* agent that will be specified as the actor, the successful execution of the action will then notified to the agent that is responsible of ensuring the consistence among the different agents activities (the discipline agent responsible for the *COD* agent is *Agent Society*), this will be the receiver. The *Add* act specified in the action has two arguments: the MAS where the element will be added and the element to introduce in it.

Communication Ontological Description Phase

In the Communication Ontological Description (COD) diagram we describe the agents' knowledge and their communications (considering related ontology, content language and agent interaction protocol). Some of these communications descend from the already studied agents' interactions, some others will be here introduced as a consequence of the study performed in the next phases and iterations.

Communications are among the most important aspects in agent-based software. Referring to the FIPA architecture we can see each communication as composed of speech acts [23] whose simplest form is: <*i, act (j, C)*> where *i* is the originator of the speech act, *act* is the name of the act (we can refer to it as a message), *j* is the target and *C* the semantic content. In the FIPA Agent Communication Language [22] this can be mapped as follows: *(act: sender i: receiver j: content C).* Note that speech acts (act in the example above) are grouped by the FIPA specifications [14] in several agent interaction protocols (AIPs) according to the intention they respond to.

This example is however still incomplete because it is lacking of the language and ontology specifications outside of which the content C makes no sense. We can therefore conclude that for each communication we need to specify three elements: ontology, content language and interaction protocol. While several languages and interaction protocols are standardized by FIPA, ontology that is often strictly related to the problem is to be defined in the specific application as already discussed in the previous section.

The Communication Ontology Diagram, is expressed in form of an UML class diagram including two different kind of elements: agents and communications. In Figure 12 we can see a portion of COD diagram reporting one of the communications of our case study; in this one the *COD* agent (that is a *fragment* agent) asks to the *DOD* agents the list of ontology elements that it will propose to the designer when he introduces a new communication in the diagram.

In Figure 12, each agent is reported as a yellow-filled class with the *agent* stereotype and each communication is shown as a relationship between two agents and identified by a unique name. The communication parameters are detailed in a class associated to the relationship. Each communication is described in terms of: ontology (this parameter values come from the elements of the ontology defined in the DOD phase), agent interaction protocol (AIP, the choice agent

interaction protocol is totally free), content language (we usually adopt RDF but any language, FIPA compliant or not, could be choosen).

The "*Ontology_Request*" communication between the *COD* agent (initiator) and the *DOD* agent (participant) refers to the *MAS Description* type of data (defined in the DOD diagram, Figure 11), the content language is RDF and the adopted AIP is Query (one of the standard FIPA interaction protocols). This description is both easy to understand and very complete and it allows the simultaneous definition of two related elements: the agent's knowledge and its communications specification.

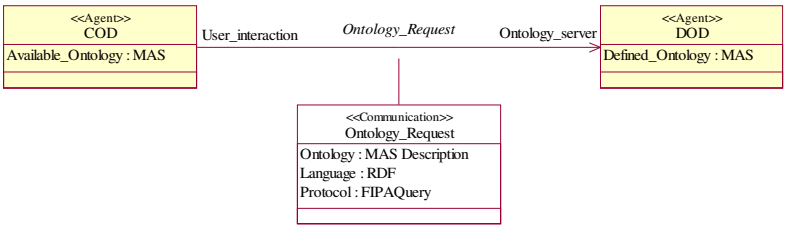

Figure 12. An example of Communication Ontological Description (COD) diagram

In designing this diagram we start from the results of the A.Id. phase. A class is introduced for each identified agent, and an association is introduced for each communication between two agents (ignoring at the moment distinctions about agents' roles). Obviously it is necessary to introduce the proper data structure (coming from the entities described in the DOD) in each agent in order to store the exchanged data.

The association line that represents each communication is drawn from the initiator of the conversation to the other agent as can be deducted from the description of their interaction performed in the RId phase.

An important piece of experience we achieved during the almost four years of designing with PASSI2 is that being our agents conceived to be communication intensive, each one of them could be related to several similar (i.e. with common elements like the ontology or content language) communications. There we can see that two different communications are just the specialization of a more general one (*GetMASElements*) and this offers the opportunity of improving the design of the *COD* agent by unifying the management of these conversations in one unique task.

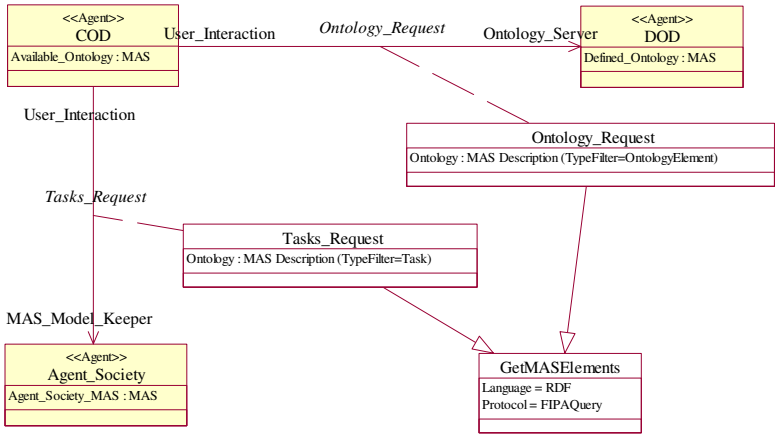

Figure 13. It is often possible to enhance the quality of the design by introducing inheritance among different communications.

Roles Description Phase

The aim of this phase consists in modeling the lifecycle of each agent, looking at the roles it can play, the collaboration it needs, and the communications in which it participates.

In the R.D. diagram we introduce all the rules of the society (organizational rules, [5]), laws of the society and the domain in which the agent operates (e.g. trade laws) and the behavioral laws considered by Newell in his "social level" [16]; these rules could be expressed in plain text or OCL in order to have a more precise, formal description.

As already said, we define a role as a portion of the social behavior of an agent that is characterized by some specificity such as a set of attributes (for example responsibilities, permissions, activities, and protocols) or providing a functionality/service. Most commonly roles we identify are devoted to provide services, share resources or achieving a goal (this is always related to ensuring the fulfillment of the functionalities that can be deducted from the use cases assigned to the agent).

The Role Description diagram (Figure 14) is a class diagram where roles are classes grouped in packages representing agents. Roles can be connected by relationships representing changes of role, by dependencies for a service or the availability of a resource and by 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.

Dependencies in PASSI2 are similar to those in i* [4]. In i*, the Strategic Dependency Model is a dependency graph, in which each node represents an actor, and each arc represents a directed dependency from one actor to the other.

Dependencies are a direct consequence of MAS cooperation, but they do not always hold when a MAS runs. Agents are autonomous and could therefore refuse to provide a requested service. For this reason, the designer needs a schema in which it is possible to analyze these dependencies and, if necessary, provide alternative ways to achieve the goal.

We consider the following kinds of dependency:

- Service dependency – An agent depends on another to accomplish a goal or perform an activity. The other agent may provide or deny the required service. This dependency unifies the goal and task dependencies defined in i*.
- Resource dependency - An actor depends on another for the availability of an entity. This is the same as i*.

Differently from i* where a specific diagram I used to represent dependencies, in PASSI2 we introduce dependencies into the RD diagram. However, we need to introduce some further syntax and stylistic conventions as follows:

- Classes represent roles.
- Methods of these classes correspond to tasks of the agent involved in pursuing the services and functionalities that under the responsibility of the role.
- Roles of the same agent are grouped in a package that is named as the agent.
- Communications among roles of different agents are shown with a solid line (i.e. a UML association), directed and named as already defined in the COD diagram.
- A change of role by an agent is shown by a dashed line from the old role to the new one with the label '[role changed]'. We represent the change of role as a dependency relationship because we want to represent the dependency of the second role from the first for the execution of some actions or the realization of a condition. Sometime the trigger condition is not explicitly generated by the first role but its precedent appearance in some scenario justifies the consideration that it is necessary to prepare the situation that allows the second role to start.
- A service dependency is shown by a dashed line (i.e. a UML dependency), with the '*service*' stereotype. The direction is towards the 'server' role.

- A resource dependency is shown by a dashed line (i.e. a UML dependency) with the '*resource*' stereotype. The direction is towards the role that owns the resource.

We can introduce external actors (representing external or not agent-based applications) in order to represent the agents interactions with other systems. In this case interactions can occur using communications (this is the case of agent-based external systems) or several different ways (remote call procedure, web services invocation …).

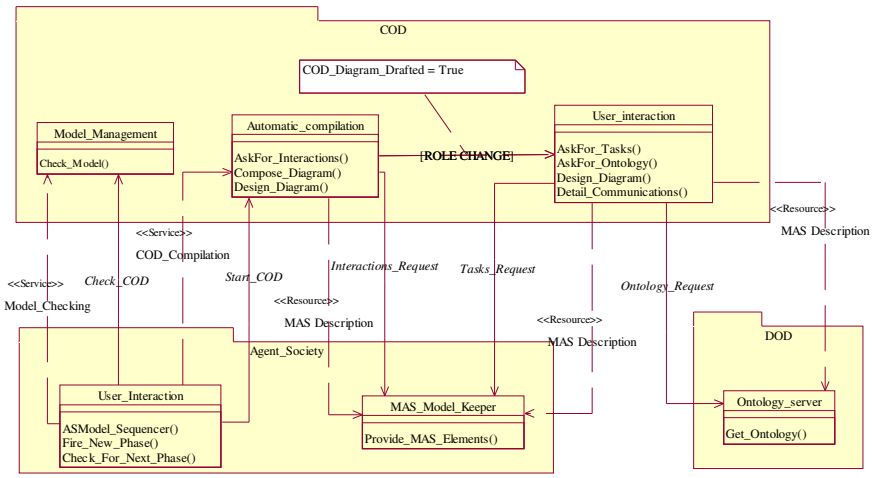

Figure 14. A Role Description (RD) diagram

In Figure 14 we can see a portion of RD diagram representing the roles, agents and communications involved in the scenario already presented in the Roles Identification activity; the *Agent_Society* agent (*User_Interaction* role) requires the *COD_Compilation* service from the *COD* agent (*Automatic_Compilation* role) with the *Start_Compostition* communication. In this last agent the *Compose_Diagram* task receives the communication as prescribed by the T.Sp. diagram (Figure 7); then the agent asks for the list of interactions among agents to the *Agent_Society* agent (each fragment level agent can interact with same level agents and its discipline agent, therefore the *COD* agent cannot acquire this information directly from the *RId* agent but it asks it to the *Agent_Society* agent that either has the required list or it looks for this data from other same level agents). Once the *Automatic_Compilation* role of the *COD* agent has completed its duty (compiling the COD diagram by using information already present in the design, i.e. providing the *COD_Compilation* service), then a change of role occurs and the agent starts to play the *User_Interaction* role. During this phase, the agent waits for user inputs (clicks on communications reported in the diagram) and then shows the form that the designer uses to introduce some communication parameters.

Some guidelines can be enumerated for this diagram:

- One class is drawn for each role of the same agent class as identified in the R.Id. diagrams.
- For each communication of the R.Id. sequence diagrams (or similarly the COD diagram) the task that is devoted to deal with that communication is introduced in the proper role. This should be in accordance with the Task Specification diagram already designed for the agent
- The role from which the first message is sent needs a communication task in order to ask for the service/resource to the other agent.
- All the tasks of the previous two steps should also be present in the T.Sp. diagrams. There we can find other tasks not directly related to communications handling. These must be introduced

in the right role accordingly to their contribution to the agent behavior. At the end, each task of the agent in the T.Sp. phase should be present in at least one role.

- If an agent can play several different roles, the series of changes can become complex and in such cases, a state diagram can be helpful in describing it.

The R.D. diagram presents information that is already in the design but which is now assembled from a different viewpoint. This is helpful in understanding and well defining services that are the most important outcome of this phases. Because of the specific nature of this diagram, it can be largely compiled by the supporting tool and the designer only needs to position the tasks in the proper role, define services and resources, model constraints for role changes. The composition of the Services description document (that is a text document) is performed as the last activity of this phase and it is left to the system.

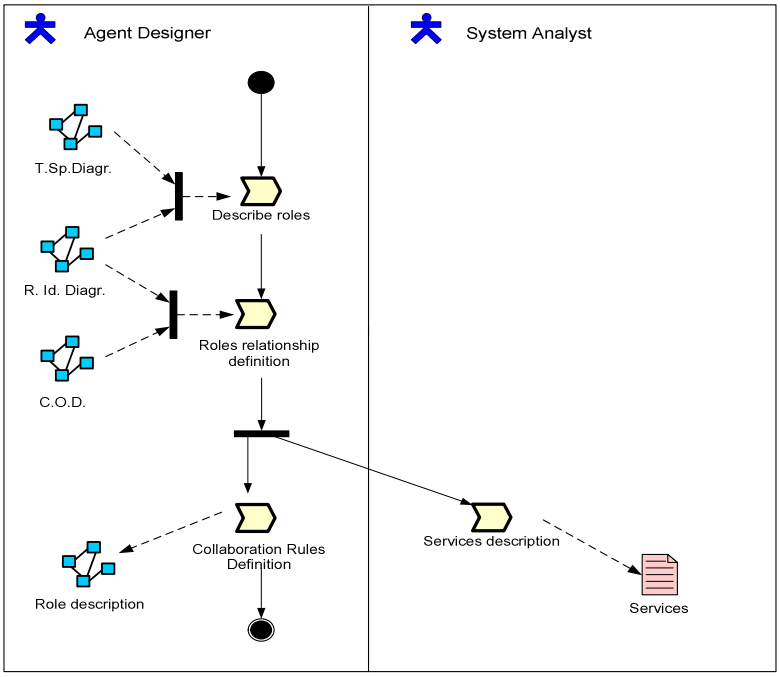

Figure 15. A SPEM activity diagram reporting the Role Description part of the PASSI2 process

1.1.1.1 Service Specification

In the RD diagram we model two different types of dependencies (resource and service); this descend from the different uses that we think an agent could do of the domain ontology elements; an agent can assert its belief (by formulating a predicate) or perform some action on domain concepts. We make a precise parallelism between these ontology elements and RD dependencies (resource and services). According to our point of view, if an agent needs to access a resource (a resource is here seen as an intangible entity that can be acquired - also using sensors - shared, or produced by agents), than it will ask to the resource owner of expressing its belief about it. Besides, if an agent desires that some action is performed on its domain (represented as a set of ontology concepts in its knowledge), it could ask to another of doing it, thus introducing the idea that the second agent is providing a service to the first. The duality between resource and service is solved in their implementation by thinking that providing access to an information (a typical resource access situation) corresponds to enabling a 'resource sharing' kind of service.

Description of services in the Service document is done by adopting OWL-S [40][41] although we often limit the number of parameters used to describe the service if the system is

closed (and therefore all the information prescribed by an OWL-S service specification is not necessary).

In this description, a specific importance maintain (using OWL-S labels) the `Service_Provider` and `Service_Requester` that are directly related to the actor and receiver entries of actions described in the DOD diagram (see 0).

Each service is identified by the `serviceName` that is also used by the agent to register the service in the platform yellow pages. This is important since we suppose that agents will acquire the address of their converser only by searching the platform yellow pages and never supposing an a-priori knowledge of their name or contact address.

We use service `precondition` to model some rules of agent society. Suppose for instance that the *COD* agent wants to get the list of identified agents in a specific project. We could imagine a scenario where it asks it directly to the *AID* agent (that is responsible for the phase where agents are identified and associated to use cases) but the *AID* agent would refuse this information because of the hierarchical organization adopted in this agent society. Each fragment agent, in fact, will grant services only to agents belonging to the same discipline-team of agents; here a discipline-team is seen as a subset of the agent society collecting: (i) fragment-level agents belonging to the same discipline and (ii) the discipline-level agent that is responsible for that discipline (for instance *System Requirements*, *Agent Society*…). The service precondition could ensure the observation of this rule if each agent registers itself (in the platform white pages) as a member of a specific discipline-team of agents and this value is included in the service input parameters.

Finally, the `serviceCategory` attribute is used to distinguish a properly named service from a resource sharing one (see discussion above). Other commonly used attributes are: `input`, `output`, and `effect` whose meaning is obvious and will not be discussed here.

Multi-Agent Structure Definition Phase

The aim of this phase consists in representing the agent society structure; we use one class diagram to show the whole MAS (of course, several diagrams could be used if the number of classes requires this), each agent is depicted using a class and detailed in terms of its knowledge and tasks.

This diagram represents the structure of the agents in a simple compact form that we found very useful to comprehend the MAS structure at a glance and to describe architectural choices (like for instance a multi-layer solution).

The contribution of this phase to the MAS Meta-model instantiation is mainly related to the iteration performed with the following MABD (Multi-Agent Behavior Description) phase (see Figure 18. The Agent Implementation Model activities and resulting artifacts. Iterating between these two phases the designer, in the MASD diagram, defines each agent in terms of the knowledge it needs to deal with its duties and the tasks representing its operational abilities. Each agent is connected to another using a relationship if a communication is supposed to happen between the two.

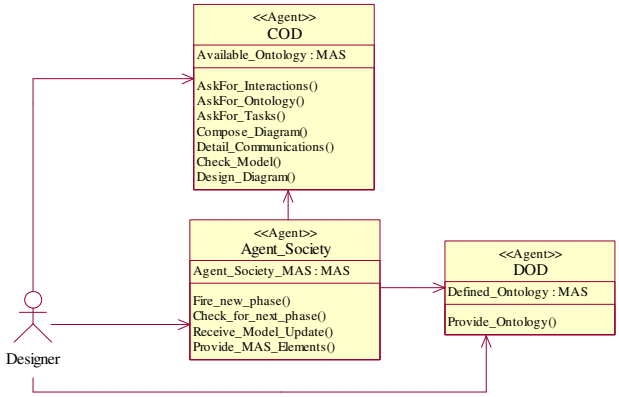

Figure 16. An example of MASD (Multi-Agent Structure Definition) diagram

In Figure 16 we can see a portion of the MASD representing the agent society for the proposed case study. This diagram differs from common UML class diagrams only for the presence of actors. We think that in a social representation of the system that is also a 'personified' view of agents it is important to represent the relationships of the agents with both users and external agents because a social representation could not be thought to be complete without explicitly considering the external elements that interacts with the system and often determine its collective behavior.

In order to complete this phase, the designer should access the initial system description performed in the ASE (Agent Structure Exploration) phase. This corresponds to start from the analysis results and then enriching them with considerations coming from the social model. In this diagram, agents coming from the ASE diagram are detailed in terms of the knowledge they need in order to cope with their social relationships (communications coming from the COD phase), and the tasks they use to accomplish their functionalities (some of these already identified in the ASE phase, some others now introduced while refining the agent).

It is also possible to introduce new agents in order to allow a better decomposition of the system from a (social) architectural point of view. A frequent example of that consists in an agent that is supposed to interact with an user via a GUI and then performs some kind of (heavy) computational task (or similarly a complex series of queries on a knowledge base/ database). If non-functional requirements prescribe that user interface should be done using a small device like a cellular phone or a PDA, the heavy-duty agent of the initial hypothesis is not a good solution and therefore it is likely to be decomposed in two (or more) different agents the first of which is a light-weighted GUI agent that can be run on strongly constrained devices.

Multi-Agent Behavior Description Phase

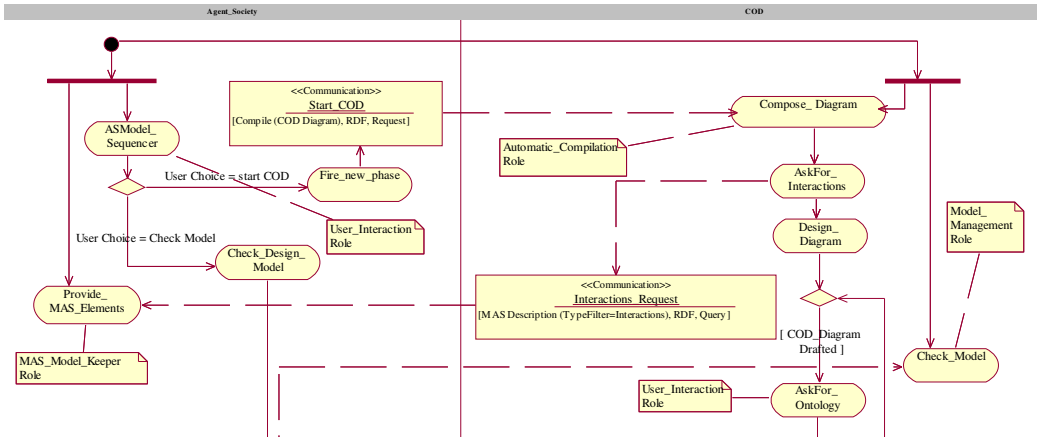

Figure 17. A portion of Multi-Agent Behavior Description (MABD) diagram

The aim of this phase consists in designing the agents' life from the social point of view. This mainly means updating the initial plan of each agent explicitly considering its social relationships (actuated using communications); in so doing the designer composes the tasks each agent owns in order to allow the pursuing of its objectives (in terms of the functionalities it has to accomplish according to what specified in the Agent Identification phase and services it can provide to the others as defined in the Role Description phase).

More in details, in this phase the designer draws one or more activity diagrams (if the system is very large) where the agents' behavior is expressed in terms of sequence of tasks and communications. Each diagram can be regarded as a refinement of the initial agent specification drafted during the TSp (Task Specification) phase (see subsection 0), but designed under a different perspective, the social one; while in the TSp diagram we were trying to define a first hypothesis of the agent plan and we were looking at an agent a time, in the MABD diagram, we are now considering the whole agent society, we are aware of the agent social relationships (communications), we know much more about the same agents' structure (in terms of behaviors and roles played) and we can therefore prepare a more complete schema of the agents' society life.

From the MAS meta-model point of view, as already reported in the artifacts structure of the Agent Society Model (Figure 9), this diagram mainly relates agents (through communications) among them and their tasks to the communications they participate. The MABD notation prescribes that the diagram is divided in swimlanes, (each swimlane representing an agent) and each activity inside them represents one of the agent's tasks. These tasks are related by refining the initial TSp plan and also considering the roles played by the agents (see RD diagram, subsection 0). The designer obtains from the RD diagram information about the roles that the agent executes in parallel (they originate a fork in the MABD diagram) and/or the changes of roles that being characterized by a logical condition introduce a decision point in the MABD; this phase is performed also considering the results of the COD (Communication Ontology Description) phase (the list of communications among agents and their details in terms of ontology, content language and agent interaction protocol) and the MASD (Multi-Agent Structure Definition) phase (the list of tasks of each agent).

The dependency of MABD and MASD phases is particularly important, from the process point of view, because they are supposed to be performed iteratively in order to introduce new structural elements (tasks) each time they are, thus originating a local iteration that allows the refinement of the multi-agent sub-model in both the structural and behavioral aspects.

The specific notation of a MAD diagram includes that transitions either represent events (e.g. a communication, represented by an object flow) or a control flow (from one task to the other). Moreover, since the syntax of UML activity diagrams already supports the representation of

concurrency and synchronization, unlike DeLoach [1], we do not need to introduce in PASSI2 a specific diagram for concurrency, besides we suppose that each agent is executed concurrently to the others (even if some of these agents can be started later by the others) and in the same agent several roles can be concurrently active.

As regards the example of MABD diagram reported in Figure 17, we can see that at system start, both the *Agent_Society* and *COD* agent are created. The *Agent_Society* agent from the beginning plays two different roles as already specified in the RD phase (see Figure 14, the *User_Interaction* role is devoted to guiding the designer in performing the process using the correct sequence of phases, and the *MAS_Model_Keeper,* that is responsible for providing information about the already designed parts of the system to fragment-level agents such as the *COD*); the first step in managing the design process is done by the *ASModel_Sequencer* task that accordingly to user's requests can fire the next phase (task *Fire_new_phase*), check the model for errors or inconsistencies (task *Check_Design_Model*) and so on. To fire the COD phase, the *Agent_Society* agent starts a communication (with the Request interaction protocol) and it asks to the *COD* agent of beginning the composition of the diagram that is under its responsibility.

Agent Implementation Model

In the Agent Implementation Model, the agent society defined in the previous models and phases is seen as a specification for the implementation of a set of agents that should be now designed at the implementation level of details, then coded, deployed and finally tested.

This model fills the gap between the agency level of abstraction where conceptual entities (agents) concur in define an abstract solution to the problem and the implementation domain where agents are seen as pieces of software (they can be resembled to the component granularity of an object-oriented program) and as such should be implemented. It is worth to note that in this release of the PASSI2 process, we mainly think at an object-oriented implementation of the solution. This does not contradict all the above arguments since it should be accepted as a matter of fact that existing proposals of agent languages are not enough mature and/or they are still remarkably based on object-oriented languages.

Specifically dealing with the implementation platforms we usually refer to FIPA-compliant ones and among them the most known are JADE, FIPA-OS and Cybele; all of them can be considered as developing frameworks for implementing Java-based agents. Some of them include testing and debugging tools (JADE, Cybele) while some others have some level of soft real-time capabilities (Cybele, at cost of FIPA communication level compatibility loss).

The activities and resulting artifacts of this model are reported in Figure 18. We can see that the first two phases are concerned with the design of the software agent from both the structural (SASD, Single-Agent Structure Definition) and behavioral (SABD, Single-Agent Behavior Description) points of view. Once the agent is ready for implementation, the system deployment configuration can be drawn in terms of a UML deployment diagram and the list of requirements (libraries, drivers, etc.) for the hosting platforms.

The design is now completed and (part of) the agent code can be automatically produced by reusing code solutions coming from applied patterns. We suppose that during all the development process, the designer is well aware of the advantages coming from a proper pattern-reuse practice.

Once the agent code is (manually) completed it is tested firstly at the agent level (a kind of unit testing) and then the social level (a kind of integration testing).

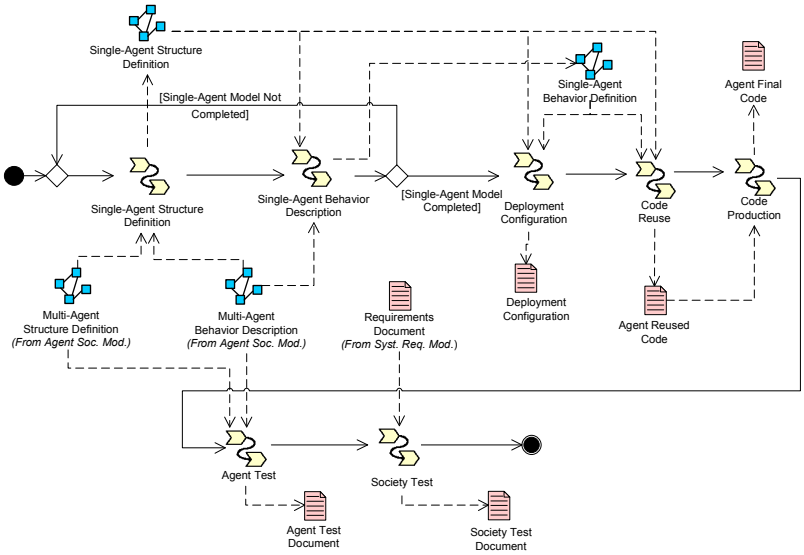

Figure 18. The Agent Implementation Model activities and resulting artifacts

The mapping between the elements of the MAS meta-model and the artefacts of this discipline are described in Figure 19. As it can be seen, the Agent Implementation Model is composed of two sets of UML diagrams (Single-Agent Structure Definition and Single-Agent Behavior Description) that describe the software solution and five text documents (the Deployment Configuration including an UML deployment diagram, two code documents and two test plan documents).

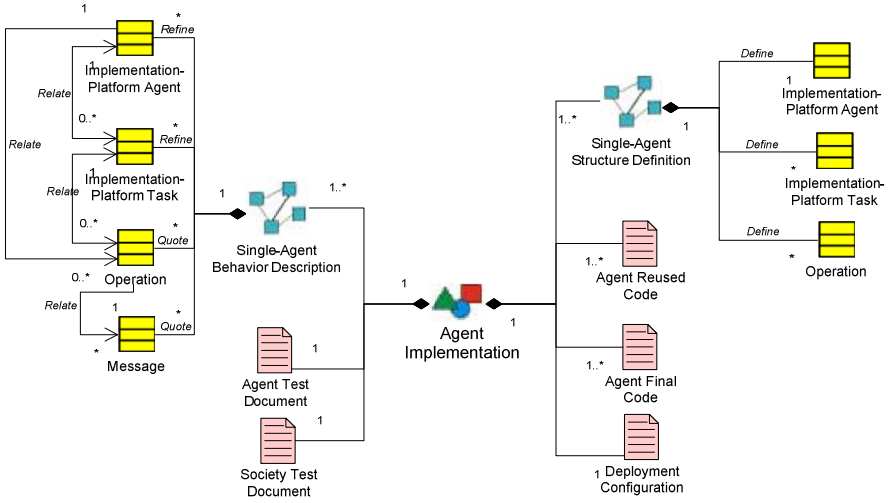

Figure 19. The artifacts structure for the Agent Implementation Model

In the next subsections we will detail the different phases of this model.

Single-Agent Structure Definition diagrams

In this diagram we address the internal structure of the classes composing the agents. We produce one diagram for each agent, in which we introduce the agent class and its tasks as inner classes.

We introduce all the methods that are needed in the different classes that have not been identified previously. These include constructors and the *shutdown* method required by the

FIPAOS environment. The tasks devoted to exchanging information need specific methods to deal with the communication events. (E.g. the *handleAcceptProposal* method in the *RegisterMtg* task, is invoked when the other agent responds with an accept-proposal communicative act to the request of registering a meeting.)

At this level of detail we have now described the structure of the software (classes, methods, attributes) in sufficient detail to implement it almost mechanically. The classes produced by following the steps described above are precisely the classes that need to be implemented in the agent coding language. What is still lacking, of course, is the description of the methods presented in these diagrams.

Single-Agent Behavior Description diagrams

It is an activity diagram, a refinement of the TSP diag. One different activity diagram for each agent. I have several swimlanes, one for each task of the agent + one for the agent itself + one for external interacting agents.

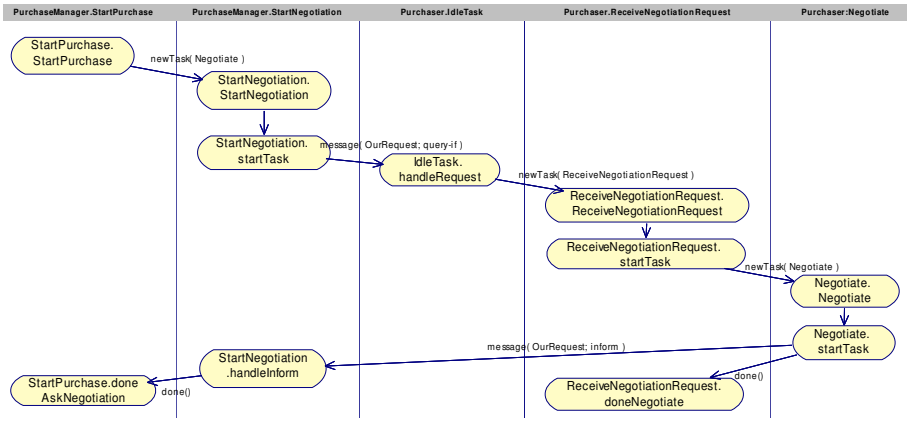

Figure 20. Single-Agent Behavior Description diagrams

This is a conventional phase for describing the implementation of the methods introduced in the (S)ASD diagrams. We can choose the any effective way to describe the specific method.

Deployment Configuration Phase

the configuration for the deployment platform can be described; this includes a diagram of the initial agents deployment and the list of modules that are necessary for agent execution (mathematical libraries, hardware controlling drivers and so on), the specification of their paths and all other specific constraints on the platform that will host the agent. It should be noted that being PASSI2 agents mobile, one agent can potentially move to several different platforms and therefore related specifications should be ensured in more than on host.

This is one of the key elements of the evolution between PASSI2 and the previous methodology developed by one of the authors (AODPU, [9]) more specifically for robotics applications. It is the response to the necessity of detailing the position of the agents in distributed systems or in mobile-agents contexts.

The deployment configuration diagram describes where the agents are located and which different elaborating units need to communicate in order to permit the communications among the

agents. As usual, elaborating units are shown as 3-D boxes (see Figure 21). Agents are shown as components; their name is in the form *agent-name: agent-class*. Communications among agents are represented by dashed lines with the *communicate* stereotype, directed as in the R.D. diagram. For each communication described in the R.D. diagram occurring between agents in different elaborating units, a dashed line is drawn. The receiving agent has an interface to show that it is capable of dealing with that communication. (I.e. It understands the protocol used.) An extension of the UML syntax is used in order to deal with mobile agents moving from one computer to another. A dashed line with the *move_to* stereotype represents it.

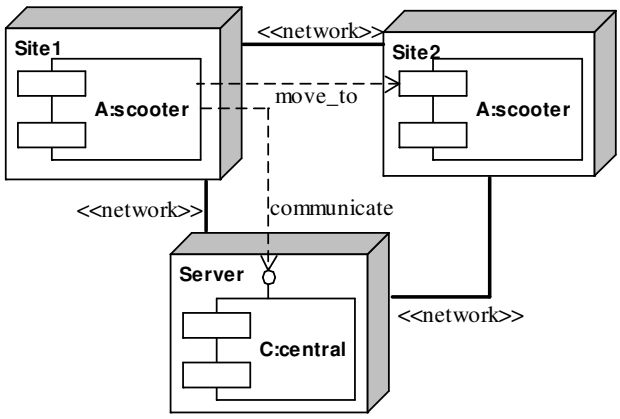

Figure 21. An example of D.C. diagram. The scooter agent moves from one node to another.

In this diagram it is also possible to specify the hardware devices used by the agents (sensors and effectors) and the modes of communication among agents in different elaborating units (traditional/wireless networks, for example). If two agents in different elaboration nodes need to communicate (as stated in the previous phases of the design), a path of connection should be provided between the two nodes.

These constraints about the connections could also be dynamic. In fact, if agent A needs to communicate with agent C, but moves across the network, we need to introduce the connection constraints as dependent on agent A's position. We introduce an OCL constraint in all the needed connections for this specific purpose.

Code Reuse Phase

Several studies have been carried on in the field of patterns for MAS and different approaches have been applied. In [2] we can find a classification of patterns for agent systems including three main categories:

- Traveling patterns (dealing with the movement capabilities of agents),
- Task patterns (dealing with the tasks that agents can perform),
- Interaction patterns (dealing with communications among the agents).

Another classification can be found in [3]. It is composed of seven levels (mobility, translation, collaboration, actions, reasoning, beliefs, sensory).

In both classification schemes we find common elements: the importance of the agent mobility, actions performed by agents, and agent collaboration or interaction. Nevertheless, the second classification is more detailed, and there we find patterns related specifically to sensors. This is a very important issue in MAS operating in the real world, such as we have in robotics applications.

Indeed all patterns are really useful only if they are well documented and obviously versatile. For this reason we have focused our work first in the production of highly reusable patterns and

then in their documentation in order to obtain a quick identification of the best pattern for a specific issue.

In the Code Reuse Phase, we try to reuse existing patterns of agents and tasks. It is not correct to talk of patterns of code only, because the process of reuse takes place in the design CASE tool where the designer looks at diagrams detailing the library of patterns, not at their code directly. Our patterns therefore are not only pieces of code. They are also pieces of design (of agents and tasks) that can be reused to implement new systems. At the programming level, the designer is too deeply involved in solving the details of the implementation of the various agents and does not have a sufficiently complete view of the system.

The best environment to try to reuse patterns is the design environment. We have used a commercial UML CASE tool that has proven very versatile, thanks to the binding of the design elements (classes, methods, attributes…) to the code language. We have therefore produced a series of pieces of reusable code that are documented with their (M)ABD and (S)ASD diagrams. In the first diagram we describe the behavior of the pattern through the sequence of events and methods implemented while in the (S)ASD we have a structural description of it in form of a class or a group of classes (for example an agent with its *IdleTask* and communication tasks for some protocols).

We have found that in our applications and with our specific implementation environment (FIPAOS), the most useful patterns are those that could be classified as 'interaction' patterns. This is due to the specific structure of FIPA language that delegates a specific task for each specific communication. Each time an agent needs to use that protocol, the pattern task could be easily reused and only the part of the code devoted to the information treatment could need of modification.

Code Production phase

This is rather a conventional phase. The programmer completes the code of the application starting from the design, the skeleton produced and the patterns reused.

Agent Test

This phase is devoted to verifying the single behavior with regards to the original requirements of the system solved by the specific agent.

Society Test

In this phase the validation of the correct interaction of the agents is performed, in order to verify that they actually concur in solving problems that need cooperation. This test is done in the most real situation that can be simulated in the development environment.

REFERENCES

[1] DeLoach, S.A., Wood, M.F., and Sparkman, C.H. Multiagent Systems Engineering. International Journal on Software Engineering and Knowledge Engineering 11, 3, 231-258.

[2] Aridor, Y., and Lange, D. B. Agent Design Patterns: Elements of Agent Application Design. In Proc. of the Second International Conference on Autonomous Agents (Minneapolis, May 1998), 108–115.

[3] Kendall, E. A., Krishna, P. V. M., Pathak C. V. and Suresh C. B. Patterns of intelligent and mobile agents. In Proc. of the Second International Conference on Autonomous Agents, (Minneapolis, May 1998), 92–99.

[4] Yu, E., Liu, L. Modelling Trust in the i* Strategic Actors Framework. Proc. of the 3rd Workshop on Deception, Fraud and Trust in Agent Societies at Agents2000 (Barcelona, Catalonia, Spain, June 2000).

[5] F. Zambonelli, N. Jennings, M. Wooldridge. Organizational Rules as an Abstraction for the Analysis and Design of Multi-agent Systems. Journal of Knowledge and Software Engineering, 2001, 11, 3, 303-328.

[6] Jacobson, I., Booch, G., Rumbaugh, J. The Unified Process. IEEE Software (May/June 1999).

[7] Jacobson, I., Christerson, M., Jonsson, P., and Overgaard, G. Object-Oriented Software Engineering: A Use Case Driven Approach. Addison-Wesley (1992).

[8] Chella, A., Cossentino, M., and Lo Faso, U. Applying UML use case diagrams to agents representation. Proc. of AI*IA 2000 Conference. (Milan, Italy, Sept. 2000).

[9] Chella, A., Cossentino, M., and Lo Faso, U. Designing agent-based systems with UML in Proc. of ISRA'2000 (Monterrey, Mexico, Nov. 2000).

[10] O'Brien P., and Nicol R. FIPA - Towards a Standard for Software Agents. BT Technology Journal, 16,3(1998),51-59.

[11] Poslad S., Buckle P. and Hadingham R. The FIPA-OS Agent Platform: Open Source for Open Standards. Proc. of the 5th International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agents (Manchester,UK, April 2000), 355-368.

[12] Cranefield, S., and Purvis, M. UML as an ontology modeling language. Proc. of the Workshop on Intelligent Information Integration, IJCAI-99 (Stockholm, Sweden, July 1999).

[13] F. Bergenti,A. Poggi. Exploiting UML in the design of multi –agent systems. ESAW Worshop at ECAI 2000 (Berlin, Germany, August 2000).

[14] FIPA Communicative Act Library Specification. Foundation for Intelligent Physical Agents, Document FIPA00037 (2000). http://www.fipa.org/specs/fipa00037/.

[15] Odell, J., Van Dyke Parunak, H., and Bauer, B. Extending UML for Agents. AOIS Workshop at AAAI 2000 (Austin, Texas, July 2000).

[16] Newell, A. The knowledge level, Artificial Intelligence, 18 (1982) 87–127.

[17] Antón, A.I., McCracken, W.M., and Potts, C. Goal Decomposition and Scenario Analysis in Business Process Reengineering in proc. of Advanced Information System Engineering: 6th International Conference, CAiSE '94 (Utrecht, The Netherlands, June 1994) 94-104.

[18] Antón, A.I., and Potts, C. The Use of Goals to Surface Requirements for Evolving Systems, in proc. of International Conference on Software Engineering (ICSE '98), (Kyoto, Japan, April 1998), 157-166

[19] van Lamsweerde, A., Darimont, R. and Massonet, P. Goal-Directed Elaboration of Requirements for a Meeting Scheduler: Problems and Lessons Learnt in Proc. 2nd International Symposium on Requirements Engineering (RE'95) (York, UK, March 1995), 194-203

[20] Potts, C. ScenIC: A Strategy for Inquiry-Driven Requirements Determination in proc. of IEEE Fourth International Symposium on Requirements Engineering (RE'99), (Limerick, Ireland, June 1999), 58-65.

[21] Jackson, M. Problem Frames: Analyzing and structuring software development problems. Addison Wesley, 2001

[22] FIPA ACL Message Structure Specification. Foundation for Intelligent Physical Agents, Document FIPA XC00061E. http://www.fipa.org/specs/ fipa00061/XC00061E.html.

[23] Searle, J.R., Speech Acts. Cambridge University Press, 1969.

[24] Zambonelli F. Jennings N., and Wooldridge M., Developing Multiagent Systems: the Gaia Methodology, ACM Transactions on Software Engineering and Methodology, 12(3):417-470, July 2003.
[25] Adelfe
[26] P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, A. Perini. TROPOS: An Agent-Oriented Software Development Methodology. Journal of Autonomous Agents and Multi-Agent Systems. Kluwer Academic Publishers Volume 8, Issue 3, Pages 203 - 236, May 2004.
[27] SPEM
[28] S. Flake and C. Geiger and G. Lehrenfeld and W. Mueller and V. Paelke, Agent-Based Modeling for Holonic Manufacturing Systems with Fuzzy ControlS. Flake, Ch. Geiger, G. Lehrenfeld, W. Mueller, V. Paelke. Agent-Based Modeling for Holonic Manufacturing Systems with Fuzzy Control, NAFIPS'99, 8th International Conference of the North American Fuzzy Information Processing Society, New York, USA, June 10-12, 1999,
[29] IGARASHI YOSHIME , TAKATA SHIRO, NIDE NAOYUKI, MASE KENJI, FIPA + BDI Architecture = Implementation of Rational Agent, IPSJ SIGNotes Mathematical modeling and Problem Solving, 032-2000
[30] Pokahr, L. Braubach, W. Lamersdorf. Jadex: Implementing a BDI-Infrastructure for JADE Agents. EXP Journal, 3-2003.
[31] P. Busetta and Kotagiri Ramamohanarao, An Architecture for Mobile BDI Agents, in Proceeding of the 1998 ACM Symposium on Applied Computing (SAC'98), 27 February - 1 March, Atlanta, Georgia (USA), J. Carroll, G. B. Lamont, D. Oppenheim, K. M. George and B. Bryant eds., ACM Press, 1998
[32] A. S. Rao and M. P. Georgeff. BDI-agents: from theory to practice. Proceedings of the First Intl. Conference on Multiagent Systems, San Francisco, 1995.
[33] D. Kinny, A.S. Rao and M. P. Georgeff, A methodology and modeling technique for systems of BDI agents, (W. Van de Velde and J.W. Perram Eds), Agent Breaking Away: Proceedings Seventh European Workshop on Modelling Autonomous Agents in a MultiAgent World, 56-71, Springer-Verlag LNAI, vol. 1038, 1996.
[34] Brooks, R. A. (1986). A robust layered control system for a mobile robot. IEEE Journal of Robotics and Automation., RA-2, April, 14-23.
[35] Brooks, R.A., "How to build complete creatures rather than isolated cognitive simulators," in K. VanLehn (ed.), Architectures for Intelligence, pp. 225-239, Lawrence Erlbaum Assosiates, Hillsdale, NJ, 1991.
[36] OMG Unified Modeling Language Specification, Version 1.5, OMG document formal/03-03-01, March 2003.
[37] M. Wooldridge, Reasoning about Rational Agents, the MIT Press, Cambridge, (MA)
[38] Resource Description Framework. (RDF) Model and Syntax Specification. W3C Recommendation. 22-02-1999. http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/
[39] FIPA RDF Content Language Specification. Foundation for Intelligent Physical Agents, Document FIPA XC00011B (2001/08/10). http://www.fipa.org/specs/fipa00011/XC00011B.html
[40] The OWL Services Coalition. OWL-S: Semantic Markup for Web Services. Available at: http://www.daml.org/services/owl-s/1.0/
[41] Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, Katia Sycara; "Semantic Matching of Web Services Capabilities." In Proceedings of the 1st International Semantic Web Conference (ISWC2002). Sardinia, Italy, June 9-12, 2002
[42] C. Bernon, M. Cossentino, M. Gleizes, P. Turci, and F. Zambonelli. A study of some multi-agent meta-models. In Proc. of the Fifth International Workshop on Agen-Oriented Software Engineering (AOSE-2004) at The Third International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2004), New York, USA, July 2004.
[43] R. Cervenka, I. Trencansky, M. Calisti, D. Greenwood. AML: Agent Modeling Language. Toward Industry-Grade Agent-Based Modeling. Lecture Notes in Computer Science, Springer-Verlag, Volume 3382/2005: Agent-Oriented Software Engineering V: 5th International Workshop, AOSE 2004, New York, NY, USA, July 19, 2004. Revised Selected Papers.
[44] M. Cossentino. From Requirements to Code with the PASSI Methodology. In Agent Oriented Methodologies, chapter iv, pagg. 79-106. Idea Group Publishing. Hershey, PA, USA, June 2005.