Autonomous Agents and Multi-Agent Systems manuscript No.
(will be inserted by the editor)

ASPECS: an Agent-oriented Software Process for Engineering Complex Systems

How to design agent societies under a holonic perspective

Massimo Cossentino[1,2], **Nicolas Gaud**[1], **Vincent Hilaire**[1], **Stéphane Galland**[1], **Abderrafiâa Koukam**[1]

[1] Multiagent Systems Group,
System and Transport Laboratory,
University of Technology of Belfort Montbéliard,
90010 Belfort cedex, France.
e-mail: {massimo.cossentino, nicolas.gaud, vincent.hilaire, stephane.galland, abder.koukam}@utbm.fr
http://set.utbm.fr

[2] ICAR Institute,
National Research Council,
Palermo, Italy.
http://www.pa.icar.cnr.it/~cossentino

Received: Sep 24, 2007 / Revised version: Apr 25, 2008

Abstract Holonic multiagent systems (HMAS) offers a promising software engineering approach for developing complex open software systems. However the process of building MultiAgent Systems (MAS) and HMAS is mostly different from the process of building more traditional software systems and it introduces new design and development challenges. This paper introduces an agent-oriented software process for engineering complex systems called ASPECS. ASPECS is based on a holonic organisational metamodel and provides a step-by-step guide from requirements to code allowing the modelling of a system at different levels of details using a suite of refinement methods. This paper details the entire ASPECS development process and provides a set of methodological guidelines for each process activity. A complete case study is also used to illustrate the methodology and to detail associated notations. ASPECS uses UML as a modelling language. Because of the specific needs of agents and holonic organisational design, the UML semantics and notation are used as reference points, but they have been extended.

Key words Agent Oriented Software Engineering – Software Development Process – Design Methodology – Holonic Multiagent Systems – Complex Hierarchical Systems

1 Introduction

Software systems characteristics and expectations have fundamentally changed in the past decade. They have increased both in size and complexity and are expected to be distributed, open and highly dynamic. Multiagent systems are emerging as an interesting software engineering paradigm for developing complex software systems [30, 52]. However, to deal with all aspects of complex systems MAS must deal with multiple levels of abstractions and openess which is not the case for most MAS [37].

According to Simon [46], complex systems often (if not always) exhibit a hierarchical configuration[1]. The idea that the architecture of a complex system can be explained and understood using a "hierarchical organisation structures" as presented in [15, 51]. Several metamodels and methodologies have been proposed for MAS [4]. However, most of them see agents as atomic entities. There are no intuitive or natural way to deal with hierarchical organisation structures. Considering agents as composed entities thus enabling a modelling of nested hierarchies offer way to solve this problem.

Giving this landscape, we advocate the use of holonic multiagent systems (HMAS) in which holons are agents that may be composed of agents for developing complex software systems. This paper introduces an agent-oriented software process for engineering complex systems called ASPECS. The process can be considered as an evolution of the PASSI [9] methodology for modelling HMAS and it also collects experiences about holon design coming from the RIO approach [28]. The construction of the new process has been performed according to the situational method engineering paradigm [27, 43] and the approach described in [10]. The complete description of the method adopted for building the ASPECS process is out of the scope of this paper. It is sufficient to say that the definition of the MAS metamodel adopted by the new process has been the first step and from this element all the others (activities, guidelines, workflow) have been deducted [10]. For this reason the description of each phase of ASPECS will start from the portion of MAS metamodel instantiated/refined by the phase activities.

ASPECS core is based on a metamodel defining the underlying concepts. A step-by-step guide from requirements to code allows the modelling of a system at different levels of details. Going from each level to the next consists in a refinement of the metamodel concepts.

Using whatever holonic perspective, the designer can model a system with entities of different granularities. It is then possible to recursively model subcomponents of a bigger system until the achievement of a stage where the requested tasks are manageable by atomic easy-to-implement entities. In multiagent systems,

[1] Hierarchical here is meant as a "loose" hierarchy as presented by Simon.

the vision of holons is someway closer to the one that MAS researchers have of *Recursive* or *Composed* agents. A holon constitutes a way to gather local and global, individual and collective points of view. A holon is a self-similar structure composed of holons as sub-structures and a hierarchical structure composed of holons is called a *holarchy*. A holon can be seen, depending on the level of observation, either as an autonomous "atomic" entity or as an organisation of holons (this is often called the *Janus effect* [33]). Holonic Systems have been already applied to a wide range of applications. Thus it is not surprising that a number of models and frameworks have been proposed for these systems, for instances PROSA [6] and MetaMorph [36, 45]. However, most of them are strongly attached to their domains of application and use specific agent architectures.

For a successful application and deployment of MAS, methodologies are essential [22]. Number of methodologies and metamodels with a clear organisational vision have been already proposed: metamodels like AGR [16], RIO [28], and methodologies like GAIA [52], INGENIAS [40], MESSAGE [7], and SODA [39]. Most of these methodologies recognise that the process of building MASs is radically different from the process of building more traditional software systems. In particular, they all recognise (to varying extents) the idea that a MAS can be conceived in terms of an organised society of individuals in which each agent plays specific roles and interacts with other agents [29, 52]. As pointed out by Ferber [16, 17], organisational approach offers a number of advantages and can contribute to agent-oriented software development in the following points: heterogeneity of languages, modularity, multiple possible architectures, security of applications.

The objective of the proposed work consists in trying to gather the advantages of an organisational approach and those of the holonic vision to model complex systems; the result will be a set of organisation-oriented abstractions that have been integrated into a complete methodological process called ASPECS.

The paper is organised as follows: after a quick introduction to the proposed ASPECS methodology (section 2), and an overview of existing MAS design methodologies (section 3), the complete ASPECS software process will be presented (sections 4–7). Throughout this description, related MAS metamodel elements and associated notations will also be illustrated by using a case study. Finally several conclusions are drawn in section 8.

2 A quick overview of ASPECS

ASPECS is a step-by-step requirements to code software engineering process based on a metamodel which defines the main concepts for the proposed HMAS analysis, design and development. It integrates design models and philosophies from both object and agent oriented software engineering (OOSE and AOSE) and is largely inspired by the PASSI and RIO approaches. The target scope for the proposed approach can be found in complex systems and especially hierarchical complex systems. The main vocation of ASPECS is towards the development of societies (organisations) of holonic (as well as not-holonic) multiagent systems.

2.1 ASPECS*: requirements*

Several key aspects have to be taken into account when designing a software development process: humans involved (intended audience and stakeholders), components (activities that define it), the underlying metamodel, the work products it delivers(with the associated languages used to document or implement them) and tools that may support the process.

The target audience of ASPECS is composed of researchers and skilled practitioners interested in complex systems modelling and complex software system development.

The conception of ASPECS was started from two opposite directions that constitute the background of this work: on one hand, the RIO and PASSI metamodels and processes were considered, and on the other hand the *Janus* platform has been selected to implement and deploy HMAS (the metamodel of *Janus* was largely inspired by RIO [28] and its holonic extensions [42]). After the design of the new process, *Janus* has been adapted to better fulfil some implementation-level requirements. Starting from this background, the development of the new ASPECS process started from the conception of its underlying metamodel. The first objective was thus to obtain a metamodel able to ensure transitions from the analysis to implementation phases by integrating both organisational and holonic related concepts.

After that the construction of the new process started; it has been based on concepts coming from the Situational Method Engineering [27, 43] and the work done by some of the authors in applying this paradigm to agent-oriented design methodologies [10, 12]. The specific approach prescribed the definition of the MAS metamodel underlined by the design process on the basis of the new process requirements (intended users, class of problems to be solved, available/selected implementation platforms and so on). This metamodel is then used as a reference point for identifying the activities that in the process will define the MAS metamodel elements in the proper way (and order).

The resulting ASPECS process now benefits from experiments and theoretical studies done on PASSI and Agile PASSI [8, 9]. Just like them, ASPECS is based on an iterative-incremental process, as it is for other widely accepted approaches in both the agent- and object-oriented contexts.

Concerning its work products, ASPECS uses UML as a modelling language but because of the specific needs of agent and holonic organisational design, the UML semantics and notation are used as reference points, and they have been extended (mainly with the definition of new profiles); in fact UML diagrams are often used to represent concepts that are not completely considered in UML and/or the notation have been modified to better fulfil the need of modelling goals and agents' behaviours.

2.2 ASPECS*: process*

The ASPECS process structure (in terms of process metamodel) is based on the Software Process Engineering Metamodel Specification (SPEM) [48] proposed by

the OMG. At the core of SPEM is the idea that a software development process is a collaboration between abstract active entities, called *Roles*, that perform operations, called *Activities*, on concrete, tangible entities, called *Work Products*. SPEM clearly separates reusable Method Content from its application in Processes. More precisely a process model is built out of *Process Elements*. Each *Process Element* can be specialised to describe one aspect of a software engineering process. According to this metamodel, the software process of ASPECS is based on three main levels: *Phases*, *Activities* and *Tasks*. A Phase delivers a composite work product (composed of one or more documents that can belong to different work product types), a phase is composed of a number of activities that are in turn decomposable into tasks. An Activity delivers a main work product (like a diagram or a text document) and it is composed of a number of Tasks. A task contributes to the production of a work product (usually by delivering a part of it), and it instantiates/relates/refines MAS metamodel elements.

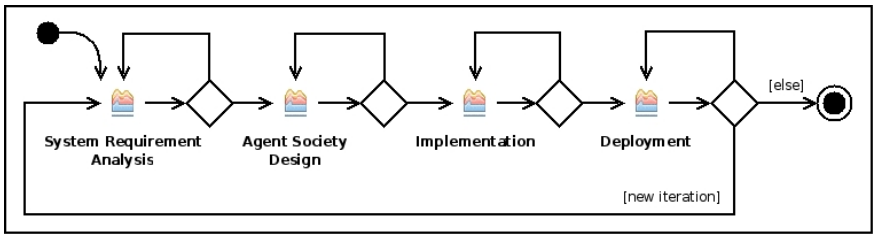

Fig. 1 Life Cycle of ASPECS

The life cycle of ASPECS consists in four phases that are briefly described below and depicted in figure 1:

1. System Requirements Analysis: the organisational description of the problem and the ontological description of the application's context.
2. Agent Society Design: the design of the agent society that is able to provide a solution to the previously described problem, and a refinement of the application context description.
3. Implementation: the description of the holon architectures, associated code production and tests.
4. Deployment: the description of the application deployment.

At the end of each phase, the designer can go back to an earlier phase if a problem or an inconsistency is detected. The description of the ASPECS software development process has been split in two different levels: the first one describes the process at the phase level, the second one reports the sub-cycle associated to each phase. Each of these phases, and their associated activities, is detailed in a separate section.Because of space concerns, each activity has been only briefly described but the interested reader may refer to the ASPECS website[2] for a complete description.

[2] http://set.utbm.fr/index.php?pge=352&lang=fr

In each activity, the related concepts of the metamodel are described. By definition a metamodel is a "model of a model", and it provides an explicit representation of the constructs and relationships needed to build specific models within a domain of interest. As it happens for the PASSI MAS meta-model, the ASPECS one introduces three domains. The first is the *problem domain* dedicated to the description of a problem independently of a specific solution and it is used in the first phase dedicated to System Requirement Analysis, that is described in section 4 (see figure 2). The second is the *agency domain* which introduces agent concepts to describe an agent solution on the basis of the elements of the problem domain, and it is related to the design of the Agent Society (see section 5, figure 13). The *solution domain* is the third and last one; it includes the elements used to implement at the code level the solution described in the agency domain. The implementation and deployment phases are based on the concepts introduced in this last domain and are detailed in section 6 (see figure 22). A complete description of the ASPECS metamodel can be found in [11].

Throughout the description of the ASPECS process, an example is used to briefly illustrate each activity. This example intends to model a simulator for the FIRA Mirosot Robot Soccer competition. This competition involves two teams of five autonomous robots playing a game similar to soccer. This is a classical example where real-time coordination is required. It constitutes a well-known benchmark for several research fields such as MAS, image processing and control.

3 Comparisons with existing Agent-Oriented Methodologies

With the increasing amount of successful applications and techniques based on the multiagent paradigm, a lot of efforts have been devoted to the definition of methods and tools for supporting agent-oriented software engineering. Most of them have been inspired by existing object-oriented software engineering techniques and adapted to deal with specific issues of agents like social concerns (organisation, role, interaction, cooperation, coordination, negotiation), complex behaviours (autonomy, mental states, beliefs, goals or tasks) and several concurrency and distribution problems. However according to Sommerville [47], all different kinds of software development processes share some fundamental activities. These include specification (definitions of software functionalities and constraints), design and implementation, validation (against customers requirements) and evolution. This section focuses only on the two initial activities and tries to give a general overview of the different existing approaches. Several proposed AOSE methodologies cover the entire software engineering process like for instance ADELFE, INGENIAS, MASE, MESSAGE, TROPOS, PASSI, and PROMETHEUS.

With the exception of formal approaches (out of scope here), specifications are generally based on the use of structured text and graphical notations. Two major perspectives are generally used: a goal-oriented or a functional-oriented (i.e. based on use-cases) one. The TROPOS methodology is based on a requirements analysis performed by means of goals, dependencies, roles, actors, positions and agents [5]. Several of these notions are similar to those adopted in ASPECS and enable to deal

with dynamic role playing. In TROPOS, they are often depicted on the same schema with the result that it is sometimes difficult to read such schemas where all concepts are on the same level of abstraction.

The MaSE methodology [13] insists upon the necessity of software tools for software engineering, specifically code generation tools. This methodology has seven sequential steps. It begins with the identification of overall goals and their structuring. Starting from goals with use cases and sequence diagrams the designer can identify roles and define them as a set of tasks. Agents are then introduced within a sort of class diagram exhibiting an extended semantic. The last step consists in precisely defining: high level interaction protocols, agent architectures and deployment diagrams. For each step a different diagram is introduced. MaSE methodology suffers from limited one-to-one agent interactions. The authors authorise a dynamic role playing relationship without providing a hint about how it can be done.

Prometheus [1] is a methodology which defines a process associated with a set of deliverables. It covers phases from specification to architectural and detailed designs. The available supporting tool verifies features at the architectural and detailed design levels and it is mainly concerned with interfaces compatibility and plans consistency.

Kendall [31] suggests the use of extended Object Oriented methodologies like design patterns and CRC. CRC are extended to Role Responsibilities and Collaborators. In [32] a seven layered architectural pattern for agents is presented. The seven layers are: mobility, translation, collaboration, actions, reasoning, beliefs and sensory. This architecture is dedicated to mobile agents. Dynamic role playing is enabled by the use of aspect-oriented programming and therefore considered only at implementation level. In the novel proposed approach dynamic role playing is considered at the analysis level too.

The Andromeda methodology [14] uses the notion of role and proposes a step by step process. It also deals with machine learning techniques for MAS and it is mainly oriented towards the design of reactive MAS. As already specified, the adoption of cognitive architectures is within the objectives of ASPECS [25].

The Anemona methodology [24] was the first complete methodology dedicated to holonic manufacturing systems. It is an extension of the INGENIAS methodology adding new activities dedicated to the identification, the design and the implementation of holon. This methodology remains strongly domain dependent and attached to specific holon architectures based on the PROSA [6] models.

Other methodologies such as ADELFE [3] or MESSAGE [7] have adopted RUP[3] [35] as their basic software life-cycle framework and use UML and AUML diagrams. ADELFE is based on an object-oriented approach and specifically deals with applications that require adaptive MAS. MESSAGE is based on an organisational approach and is dedicated to telecommunication applications. They both consider the distinction between Role and Agent as analogous to that between Interface and Object Class. Kristensen and Osterbye [34], studying the notion of role for objects, called such a vision of role as the *Filter Metaphor* and dis-

[3] Rational Unified Process

cussed the problems related to such an approach: "*This is mistaken because the filter metaphor implies that the persons*[4] *has all the properties from the outset, and we choose to see only some of them. This neglects the important meaning behind roles, that the properties are extrinsic, - the person only has them because of the role*". In ASPECS the role is emphasised as a fundamental entity spreading from requirements to implementation. It is the abstraction of a behaviour in the context defined by its organisation and it gives a status to the players who play it in the organisation itself (refer to section 4.4 for more details); each role has its own specific properties and behaviour.

In [21] MaMA-S is composed of three methodological phases: analysis and specification, design, and implementation. Recursive multiagent concepts are introduced in the second phase since the first one is reserved to system domain specialists. According to its UML metamodel which supports collaborative design MaMA-S provides models and tools to check and verify several properties of the models resulting from the two first stages.

As regards relevant modelling aspects, Bergenti and Poggi [2] suggest the use of four UML-like diagrams. These diagrams are modified in order to take into account MAS specific aspects like: conceptual ontology description, MAS architecture, interaction protocols and agent functionalities.

Other interesting modelling issues are raised in [38], where the authors extend UML for representing agent related notions. In particular, the authors insist upon the role concept and suggest the use of modified sequence diagrams to deal with roles.

4 System Requirement Analysis

System requirements analysis phase aims at providing a full description of the problem based on the concepts defined in the *Problem Domain* area of the metamodel. The metamodel is presented in figure 2. Each concept of the problem domain will be detailed in the activity where it is defined in what follows.

This phase starts with requirements analysis and stops when the resulting model is considered to be in conformity with the identified requirements or when an agreement between designers and stakeholders is reached. Functional requirements and non functional requirements are identified using classical techniques such as use cases. Each requirement is then associated to an Organisation. An Organisation is defined by a set of AbstractRoles, their Interactions and a common context defined according to an ontology. In the process, the application's context is described at first, then organisations are identified and finally decomposed into interacting smaller behaviours. Interactions in the system are described using a set of scenarios. A Scenario describes how roles interact and coordinate to satisfy goals assigned to their organisation. The behaviour of each AbstractRole is specified within a Role Plan. Such a plan describes how role goals can be achieved. The last activity is dedicated to capacity identification. The capacity element constitutes a layer between the role behaviour and the future entities which will want

[4] Person here is meant as the entity playing the role

to play this role. Basing the description of role behaviour on capacities, gives to the role more genericity and modularity.

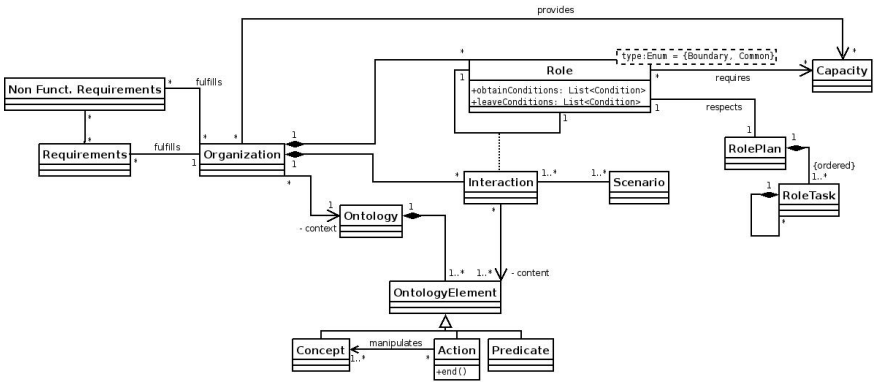

Fig. 2 The UML diagram of the ASPECS metamodel Problem Domain

The various activities involved in the System Requirement phases and the set of associated diagrams are described in figure 3. In the following subsections a description of each System Requirement Engineering activity is presented.

4.1 Domain Requirements Description (DRD)

Both the PASSI, and ASPECS, software processes are driven by requirements. Thus the starting activity deals with the functional requirements and non functional requirements analysis. Functional requirements describe the functions that the software is to execute [**?**] or the behaviour of the system in terms of interactions perceived by the user. Non functional requirements are sometimes known as constraints or quality requirements [**?**]. The global objective of the Domain Requirement Description (DRD) activity is to provide an overview of the system's functionalities. This activity thus aims at gathering needs and expectations of application stakeholders and at providing a complete description of the behaviour of the application to be developed. In the proposed approach, these requirements (both functional and non functional) should be described using the specific language of the application domain and a user perspective.

Figure 4 details the use cases associated to the development of a simulator for the FIRA Robot Soccer cup. Eight use cases and one actor have been identified, each use case has been numbered in order to facilitate explanation in the following activities. The actor represents the user of the simulator who can simulate matches and tune the strategy of each team. Simulating a match implies the simulation of two autonomous teams that can choose their own strategy and are responsible for simulating the individual robots behaviour.

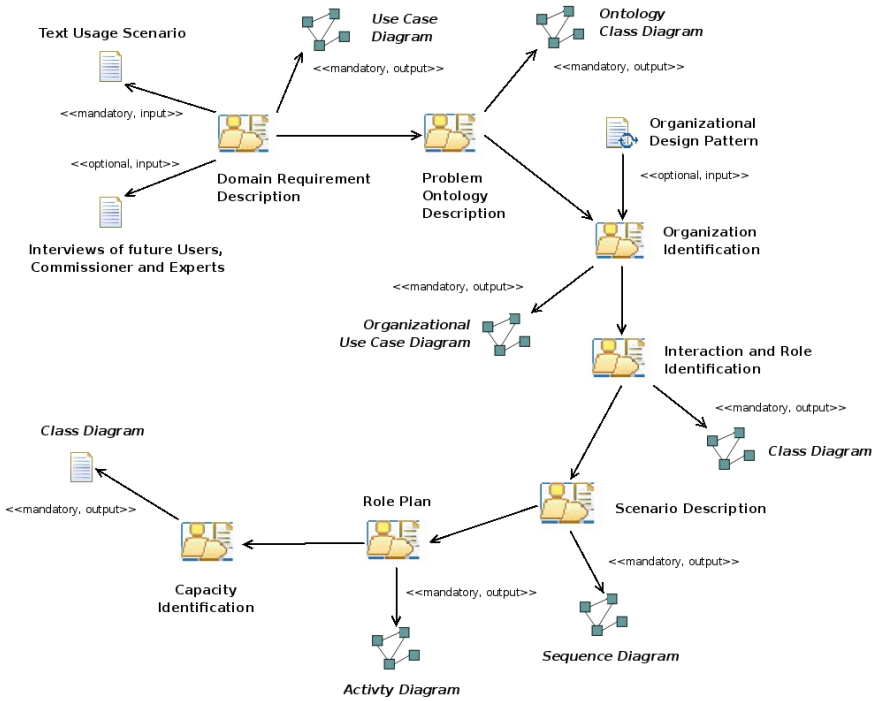

Fig. 3 Details of the System Requirements Analysis Phase

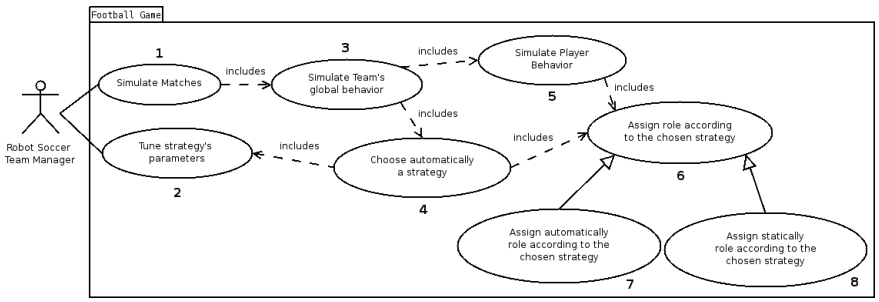

Fig. 4 FIRA Robot simulator use cases

4.2 Problem Ontology Description (POD)

The global objective of the Domain Ontology Description is to provide an overview of the problem domain. Stakeholders naturally express requirements in their own terms and with implicit knowledge of their own works [47]. Therefore the aim of this activity is deepening the understanding of the problem by complementing the usual requirements description in terms of use cases with a description of the concepts that compose the problem domain. It describes concepts used in the specific language of the application domain and users. Results of this work can sometime imply modifications in uses cases. The design of the domain ontology

occurs very earlier in our methodological process and this denotes its substantial role in the organisation identification and capacity identification activities.

Ontology associated to the domain of the FIRA Robot soccer is partially detailed in figure 5. This ontology represents knowledge about robot soccer matches such as, for example, time periods, different mobile (and thus situated) elements, rules handlin.

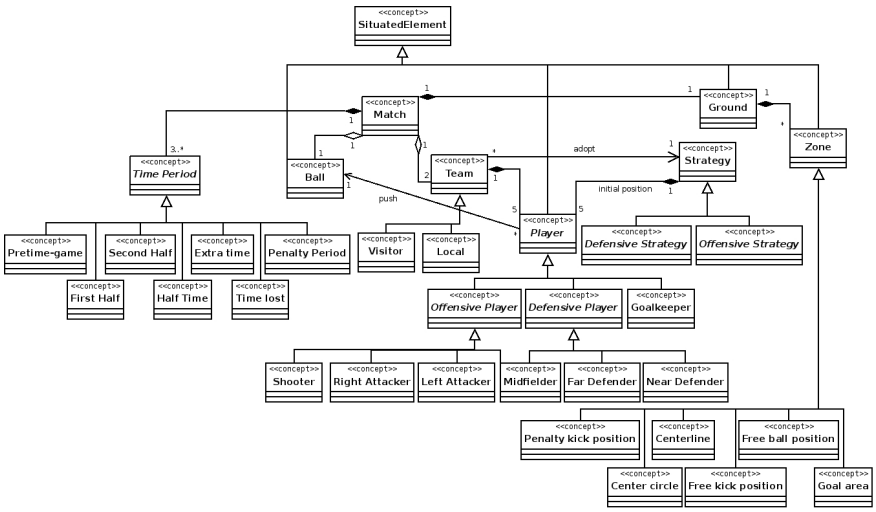

Fig. 5 Ontology of the problem domain associated to FIRA Robot soccer case study

4.3 Organisation Identification (OID)

The goal of the Organisation Identification activity is to bind each requirement to a global behaviour, embodied by an organisation. Each requirement is then associated to an unique organisation (see figure 2) in charge of fulfiling it. As already said, an organisation is defined by a set of Abstract Roles, their Interactions and a common context. The associated context is defined according to a part of the Problem Ontology, described in the previous activity. Organisation and Role identification are two key activities and probably the two most difficult ones in our process, because these two aspects are the basis of the whole methodological process and occur quite early in the workflow.

Starting from use cases presented in the DRD activity, different approaches could be used to cluster them and identify organisations. The first task, suggested by ASPECS, consists in identifying which functionalities will be designed using an agent-oriented approach and which ones with a traditional object-oriented approach.

In the FIRA soccer example, functionalities belonging to use case 2 are designed with an OO approach (see figure 4). In this situation, this use case is extracted from the agent-oriented part of the system, and it is represented as a new

actor, called *Tuning interface*. The part of the system corresponding to this use case will be designed separately using a classical object-oriented software process like RUP [35]. Concerning the other use cases, three possible partitions can be studied, each one corresponding to a different point of view on the system:

Functional approach: Three main functional areas are identified, and an organisation is associated to each one (see Fig. 6). Firstly the global system is itself associated to an organisation *Game Simulation* managing use case 1. Simulation functionalities are associated to a second organisation called *Team Simulation* that is in charge of use cases 3 and 5. And finally strategy determination is associated to an organisation called *Strategy Decision* grouping use cases 4, 6, 7 et 8.

Ontological approach: This partition is based on the hierarchical decomposition of the ontology (see Fig. 7). Uses cases referring to ontological concepts of the same level are grouped. At the top level of the hierarchy, an organisation named *Game Simulation* grouping use case 1 is defined and it is related to the highest concept in the ontology, *Match*. Then use cases 3 and 4, referring to *Team* and *Strategy* are assigned to a unique organisation and at the third level the organisation *Player Simulation* is introduced to deal with use cases 5, 6, 7 and 8, attached to sub-concepts of Player in the ontology. It is interesting to note that this approach introduces a nice hierarchy of organisations.

Multiview approach: The multiview approach consists in merging the various points of view we can have on the system (including the two previous ones). This approach respects the hierarchical nature of the system revealed by the ontological approach but it clearly separates use cases attached to different system functionalities. The last solution was preferred as granularity of functionalities and of the different levels of abstraction present in the system are respected. This case is detailed in figure 8.

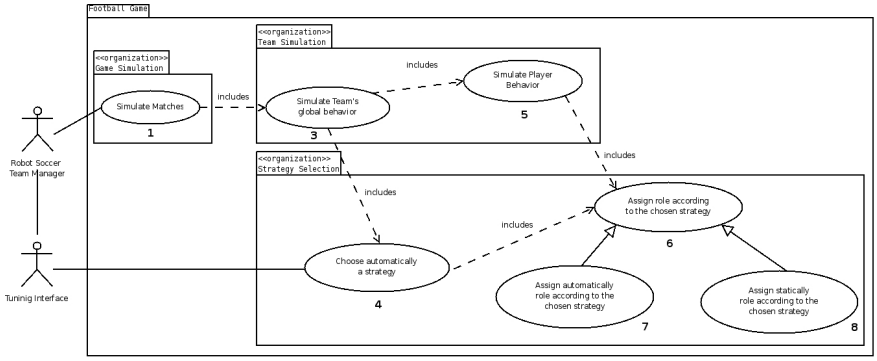

Fig. 6 Organisation identification using the functional approach

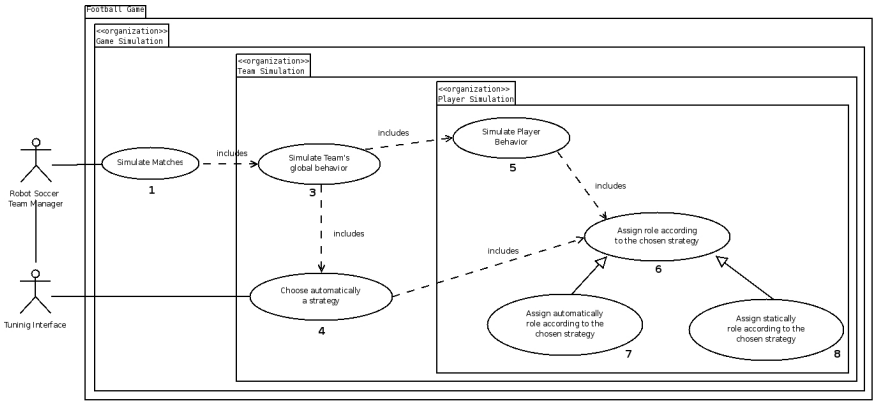

Fig. 7 Organisation identification using the ontological approach

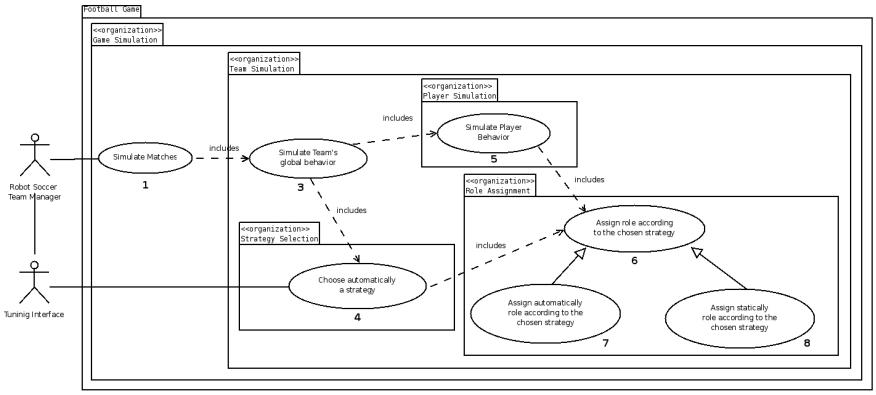

Fig. 8 Organisation identification using the multiview approach

4.4 Interactions and Role Identification (IRI)

Interactions and Role Identification (IRI) aims at decomposing a global behaviour embodied by an organisation into smaller interacting behaviours. Each of these finer grained behaviours will be represented by an AbstractRole. Thus, an AbstractRole is the abstraction of a behaviour in a certain context defined by the organisation and confers a status within this context. AbstractRoles interact according to one or more specific interaction patterns. An Interaction is composed of the event produced by a first role, perceived by a second one, and the reaction(s) produced by the second role. The sequence of events from one role to the other can be iterated several times and includes a not a priori specified number of events and participants. Interacting roles must be defined in the same organisation that provides the interaction context. The status of an AbstractRole is defined as a set of rights and obligations available to the role, and also defines the way the entity playing the role is perceived by other entities playing another role in the same organisation. Specifically, the status gives the playing entity the right to exercise its

capacities (know-how). Another important aspect is that the role (and not the individual, like an agent or an holon, which plays the role) belongs to the organisation. This means that the same individual may play various roles in the same organisation that may be perceived as different (and not necessarily related). Besides, the same individual can play different roles in other organisations.

The goal of each AbstractRole is to contribute to the fulfilment of (a part of) the requirements of the organisation it belongs to.

This activity also aims at completing the system delimitation started in the domain requirements description activity. Before trying to develop a system, the perimeter of the application has to be defined. AbstractRole is a parametrised class that can be instantiated in Common AbstractRole or Boundary AbstractRole. This last has been designed to delimit the borders of the designed system. A Boundary AbstractRole is an AbstractRole located on the boundary between the system and its outside and it is responsible for interactions happening at this border (i.e. GUI, Database, etc).

According to the previously identified hierarchy of organisations, the objective is now to decompose each organisation and precise the behavioural contribution of sub-level organisations to an upper-level organisation. In this example a top-down behavioural decomposition is used and for each level a test for the necessity of decomposition in sub-levels is done. Figure 9 describes the result of this decomposition: the various organisations and their respective contributions.

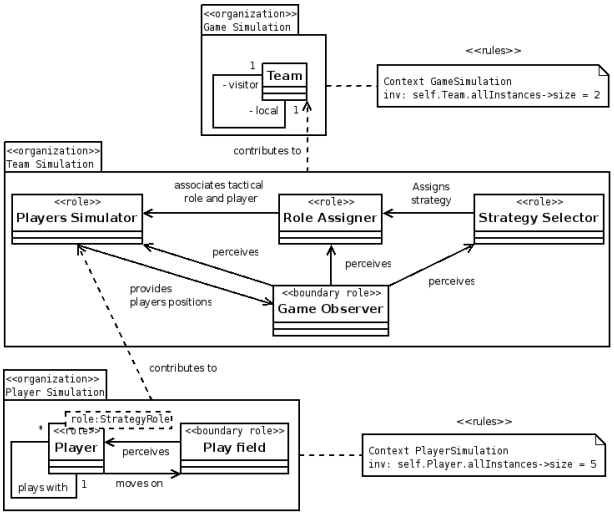

Fig. 9 Result of the Interaction and Role identification for the Robot soccer simulator example

At the top of the hierarchy, the *Game Simulation* organisation is decomposed using only one interaction and one role: *Team*. An OCL constraint is added to specify that only two instances of this role are allowed in each instance of this organisation. This role is in charge of simulating the behaviour of a robot soccer

team. Its complexity at this level is considered as too high and it will be decomposed into smaller interacting behaviours.

At the second level, the organisation *Team Simulation* is decomposed in four roles: three roles representing the contributions of previously identified sub-level organisations and a boundary role useful to provide required information about the game situation (players and ball position, score, etc). The *Strategy Selector* role is in charge of determining the best strategy to adopt according to the current situation. Each strategy corresponds to a distribution of strategy roles like *goal-keeper*, *near-defender*, *midfielder*, *shooter*, etc. among the different players. Each robot soccer team is composed of five players. Applying a strategy thus consists in assigning to each player one of the five roles defined by the strategy. The association between players and roles defined by the strategy is done by the *Role Assigner* role. The *Players Simulator* role is in charge of simulating the behaviour of the various players according to the chosen strategy. The global state of the game (players and ball positions, score, time, etc.) is maintained by the boundary role *Game Observer*. This role is also in charge of providing required perceptions to the others.

According to the work products of the organisation identification activity, each of the three first roles (*Role Assigner*, *Strategy Selector* and the *Players Simulator*) is associated to a use case and should be decomposed into sub-organisations. But at this step one can already decide that the behaviours granularity is sufficiently detailed. The *Role Assigner* and *Strategy Selector* role are considered as sufficiently detailed, the *Players Simulator* role may be decomposed. This reveals a new level in the hierarchy, and the *Player Simulation* organisation is introduced. It defines the *Player* role in charge of simulating the behaviour of a robot playing soccer according to a specific strategy role and one boundary role representing the *Play Field* and maintaining the state of the game at this level of abstraction.

4.5 Scenario Description (SD)

The goal of this activity is to describe the sequence of interactions among the roles involved in each scenario. A Scenario describes how roles interact and coordinate to satisfy goals assigned to their organisation.

Scenarios description occurs just after *OID* and *IRI* activities and at this stage it is possible to assign to each requirement an organisation and a set of interacting behaviours (enacted by involved roles). The challenge is now to describe how these different roles interact to realise the scenario.

Figure 10 describes a scenario for theTeamSimulationOrganisation. Each role of this organisation has its own lifeline. Above each lifeline a box specifies the name of the role and the name of the organisation it belongs to, separated by a colon. For example, the role Team belongs to the GameSimulation organisation of the upper level. The exchanged messages are ordered as follows. The scenario starts with Team sending an initialisation messages to all other roles. The StrategySelector decides of a strategy which is sent to the RoleAssigner which, in turn, decides of a role assignment and sends it to the PlayersSimulator. Robots are as-

signed locations on the field and the PlayersSimulator sends a ready-to-start message to the Team. The game begins when the human user starts the match. The role Team forwards the start message.

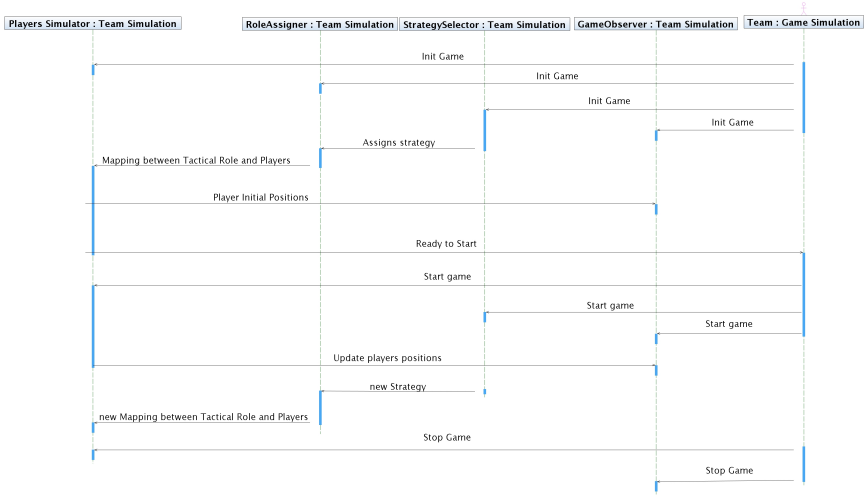

Fig. 10 Scenario of the Team Simulation Organisation

4.6 Role Plan (RP)

The goal of each AbstractRole is to contribute to (a part of) the requirements of the organisation within which it is defined. The behaviour of an AbstractRole is specified within a Role Plan. The goal of this activity is to conceive for each role a plan that could fulfil the part of the organisation requirements that have been delegated to the role under study. In this context a plan describes how a goal can be achieved. It is the description of how to combine and order interactions, external events, and RoleTasks to fulfil a (part of a) requirement (the goal). A RoleTask is the specification of a parameterised behaviour in form of a coordinated sequence of subordinate units (a RoleTask can be composed of other RoleTasks). The definition of these units can be based on capacities, required by the role.

At this step, we focus on each role in order to define a plan that may accomplish the requirements coming from the previous parts of the design. The first task in this activity consists in detailing responsibilities assigned to the currently designed role. Then for each of them, a set of RoleTasks has to be identified for accomplishing the assigned requirements. The final step consists in determining transitions between the various activities and the set of associated conditions. In a second iteration each task will be examined to be eventually decomposed and in order to determine if it requires something external to the role. If this is the case then a new capacity will be created and the role will refer to it.

The description of role plans associated to the *Team Simulation* organisation is presented in figure 11. The activity diagram is partitioned according to the different roles of the organisation, a supplementary partition line is added to represented the part of the system external to the organisation under study. Interactions with this external part are represented by adopting an event-driven perspective.

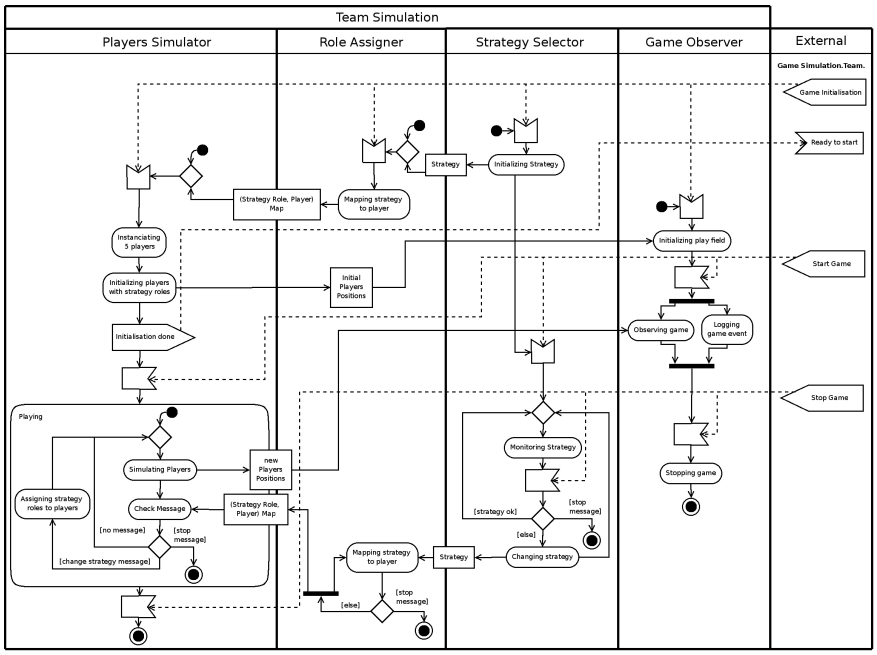

Fig. 11 Plan of the Team Simulation Organisation

4.7 Capacity Identification (CI)

The notion of capacity describes what an organisation is able to do. It has been introduced to control and exploit additional behaviours emerging from roles interactions, by considering an organisation as able to provide a capacity. Organisations used to model role interactions offer a simple way to represent how these capacities are obtained from roles. A role may require that individuals playing it have some specific capacities to properly behave as defined. An individual must know a way of realising all required capacities to play a role.

In other words, the main objective of the Capacity Identification activity (CI) is the definition of generic role behaviours by identifying which know-how a role requires from the individual that will play it; The capacity element constitutes a layer between the role behaviour and the future entities which will want to play this role. Basing the description of role behaviour on capacities gives to the role

more genericity and modularity. A complete description of the concepts of capacity and template that describes a capacity can be found in [41]. For the scope of this paper it is worth to say that the capacity is a description of what an organisation (and therefore one of its composing roles) is able to do without any kind of specification on how to do it. This means that the results prescribed by a capacity may be reached by adopting different strategies (the realisation of the capacity is a concern of the Agency Domain and will be discussed later).

For instance, if an entity's personal acquaintances influence a particular choice that is required in the role behaviour, e.g. the choice of the best partner to fulfil a task, this choice may be externalised as a capacity. Indeed there are various ways of carrying out a capacity and they depend on data which are strictly related to the entity personality (beliefs, acquaintances, etc). This is often a design choice. This function can be hardly coded in the role behaviour and imposed to the entity which plays the role. Another possibility is to externalise and let each entity free to carry it out in the way it prefers. This depends obviously on the application and the level of generality the designer wants to give to the role.

Considering the previous case study and specifically the *Team Simulation* Organisation, it is possible to note that, in this organisation the *Strategy Selector* role has to identify the best strategy for a given game situation. This choice can typically be implemented under various perspectives, even decomposed using a sub-level organisation. This can be extracted from the role behaviour by creating a specific capacity called *ChooseStrategy*. In the same way the *Game Observer* role is able to maintain the information related to state of the game (player and ball positions, score) and to check if game rules are observed. This behaviour requires the possibility to observe the game and this observation mechanism can be implemented in different ways requiring specific rights. A capacity can also be used for dealing with such a situation. Figure 12 summarises the different capacities identified in this organisation.

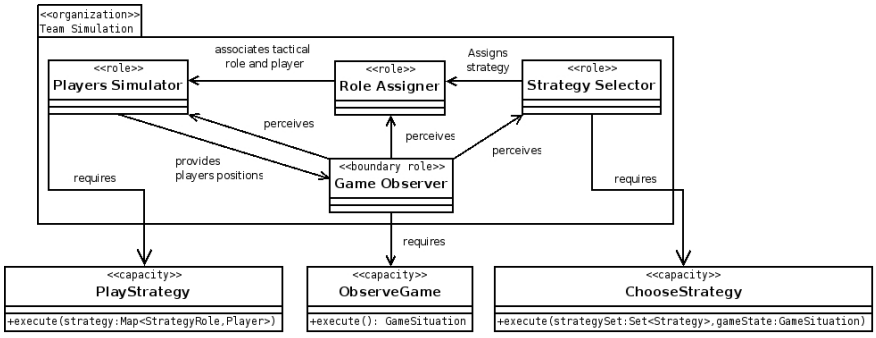

Fig. 12 The Team Simulation Organisation and its set of capacities

5 Agent Society Design

This phase aims at designing a society of agents, whose global behaviour is able to provide an effective solution to the problem described in the previous phase and to satisfy associated requirements.

At this point, the problem is modelled in terms of organisations,roles,capacities and interactions. The objective of this phase is to provide a model of the agent society involved in the solution in terms of social interactions and dependencies among entities(Holons and/or Agents)

The Agency Domain part of the HMAS metamodel is reported in Figure 13; some of its elements (Group, AgentRole, Capacity, AgentTask) are a specialisation of other elements defined in the Problem Domain (the two domains are separated by a dashed line); they constitute the backbone of our approach and are refined from one domain to the other in order to contribute to the final implementation of the system. In this phase organisations' context and the associated ontology are enriched with solution-related concepts. It is worth to note that in the proposed approach no new ontological elements have been introduced in the metamodel when passing to the agency domain because ontological representations are considered as transversal to the proposed classification in problem and agency domains. AbstractRoles are refined into AgentRoles and their behaviours are decomposed into a set of AgentTasks and AgentActions. An AgentRole interacts with the others using communications and may contribute to provide (a portion of) a service; a service can implement a capacity. Two very important elements of the MAS metamodel are newly introduced in this activity; they are Agent and Holon. An Agent is an entity which can play a set of roles defined within various organisations; these roles interact each other in the specific context provided by the agent itself. The concept of Holon is specialised from the Agent one. An holon can play several roles in different organisations and it may be composed of other holons. A composed holon (a super-holon) contains at least an instance of a holonic organisation to precise how members organise and manage the super-holon and a set (at least one) of production organisations describing how members interact and coordinate their actions to fulfil the super-holon tasks and objectives.

Details of activities and the set of associated diagrams involved in the Agent Society Design phase are described in figure 14. In the following sub-sections each activity will be detailed according to the already introduced template of description.

5.1 Solution Ontology Description (SOD)

The objective of this activity consists in refining the problem ontology described during POD by adding new concepts related to the agent-based solution and refining the existing ones. Concepts, predicates and actions of this ontology are now also intended to be used for describing information exchanged during communications among roles.

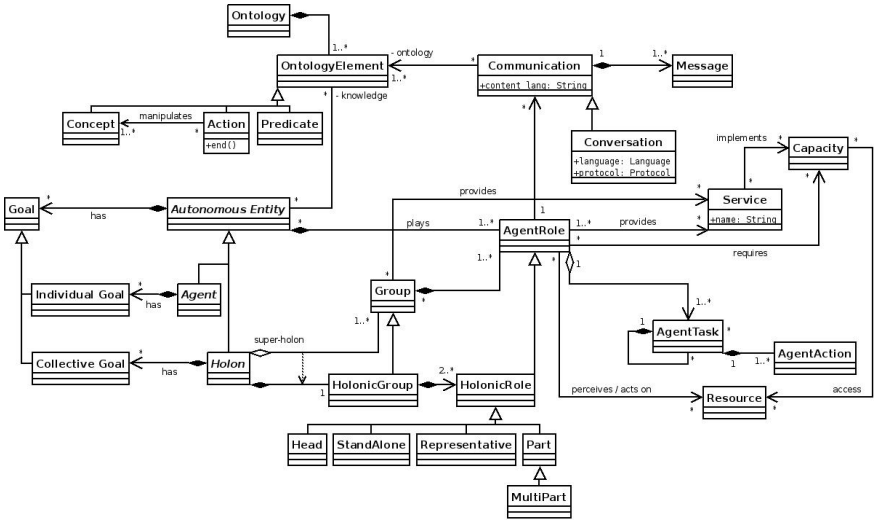

Fig. 13 The UML diagram of the ASPECS metamodel

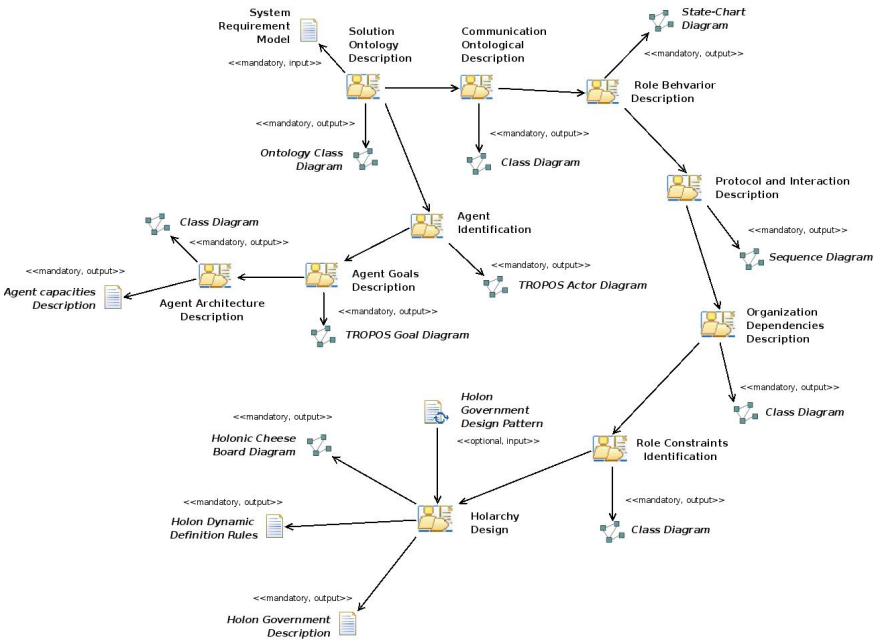

Fig. 14 The Agent Society Design Phase

The first task consists in refining existing concept descriptions and identifying new ones (related to the agency-level solution that is under development). Actions and predicates should also be added.

Figure 15 describes a part of the Solution Ontology associated to our example of the FIRA Robot Soccer Simulator.

This ontology is a refinement of the Problem Ontology previously described in the Domain Requirement Description phase. By refinement we mean that new concepts which inherit from existing ones are added. These newly introduced concepts are related to the description of elements localisation in the play field some associated actions and predicates are also defined. For instance, actions related to game observation are introduced due to the identification of the *ObserveGame* capacity in the *Team Simulation* organisation.

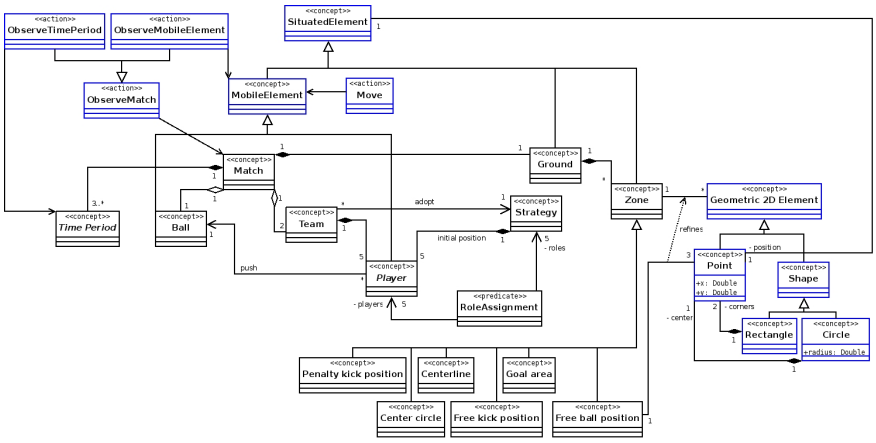

Fig. 15 Class diagram describing a part of the Solution Ontology for the Robot Soccer Simulator case study

5.2 Communication Ontological Description (COD)

This activity aims at describing communications and conversations among roles. Communications are a more refined way for interacting compared to the basic Interactions allowed to the AbstractRole. Interactions identified in the problem domain are specialised in this domain in communications. The model of role communication is based on the assumption that two roles wishing to interact, share a common ontology. This common knowledge is represented in the communication by a set of Ontology Elements. A conversation is a specialisation of a communication in which content (Language, Encoding, Ontology) and control (Protocol) of the communication are detailed. A conversation mainly consists of FIPA[5] speech acts [19, 20] and protocols. A protocol defines the sequence of expected messages.

Each set of interactions between two roles has to be clustered in one or more communications/conversations. At this stage we could regard the previous studied interactions as messages of the communication/conversation. Interactions linking

[5] Foundation for Intelligent Physical Agents: http://fipa.org

a boundary role to another boundary one are generally refined in communications (usually these interactions are mediated by the environment and therefore they are not classical message-based conversations). They can be based on events or other stimuli sent from one role to the other. Interactions between classical (i.e. non-boundary) roles are refined in conversations with a defined protocol, a referred ontology and an associated content language. As a consequence in conversations, the sequence of messages will be ruled by a proper interaction protocol. The content will be related to the exchanged information or required exhibition of a capacity through an explicit reference to the adopted ontology. This is in accordance with the FIPA specifications [19] (that we largely adopt), where conversations consist of speech acts [44]. FIPA also specifies a relevant number of interaction protocols but a new one, if necessary, can be designed as reported in section 5.4. This activity should also describe data structures required in each role to store exchanged data by adding the necessary ontological structure to roles. These structures are of course based on the elements of the solution ontology.

Let's now consider the *Team Simulation* organisation of our example. In this organisation six interactions have been identified. Two of them may be modelled as a conversation. For instance, the interaction between *Players Simulator* and *Role Assigner* may be considered as a conversation respecting the FIPA Inform Communicative Act [20], manipulating the *Strategy Role Assignment* predicate previously defined in the Solution ontology and adopting the RDF (Resource Description Framework) content language [18]. Figure 16 reports several communications and conversations of the *Team Simulation* organisation. Each conversation is represented by an association class linked to the interaction. The attributes of this class are ontology, language and protocol.

5.3 Role Behaviour Description (RBD)

This activity aims at defining the complete life-cycle of a role by integrating previously identified RoleTasks, Abstract Capacities, comunications/conversations in which it is involved and the set of (or part of) requirements it has to fulfil. AbstractRoles identified during the IRI activity are specialised here in AgentRoles. An AgentRole interacts with the others using communications. The behaviour of the role is described by a set of AgentTasks and AgentActions. AgentTask is the specialisation of the RoleTask. It is aggregated in AgentRole and contributes to provide (a portion of) an AgentRole's service. At this level of abstraction, this kind of task is no more considered atomic but it can be decomposed in finer grained AgentActions. An AgentAction is now the atomic unit of behaviour specification. An action takes a set of inputs and converts them into a set of outputs, though either or both sets may be empty. An example of the most basic AgentAction consists in invoking a capacity or requesting a service (as explained in following subsections).

A Role Behaviour Description is a refinement of the results produced by the Role Plan activity (System Requirement phase). The behaviour of each role is now described using a statechart or an activity diagram but the use of statecharts is preferred because of the intrinsic state-based nature of roles. If a role requires

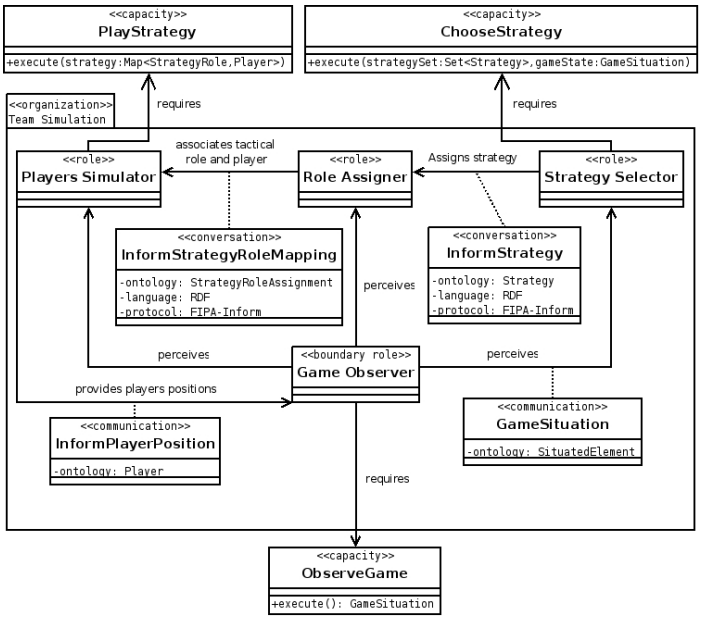

Fig. 16 Class diagram describing some of the communications/conversations designed for the Robot Soccer Simulator case study

capacities or provides services, this activity has to describe tasks and actions in which they are really used or provided. The designer describes the dynamical behaviour of the role starting from the Role Plan drawn in the previous phase and the capacities used by the role.

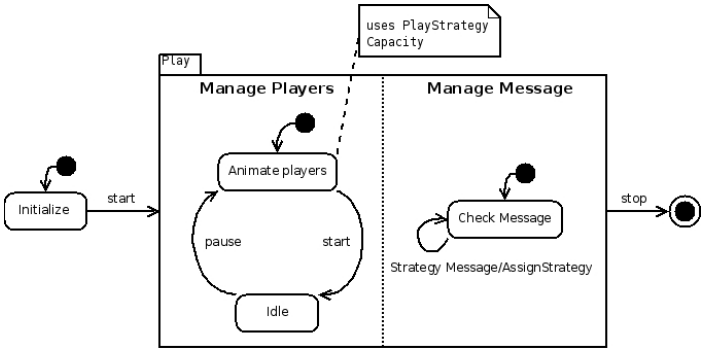

Fig. 17 Statechart of the Players Simulator role of the Team Simulation organisation

Figure 17 describes the statechart associated to the *Players Simulator* role of the *Team Simulation* organisation. It corresponds to the refinement of the behaviour described by the role plan in the previous phases. The default state is *initialise* and then after a *start* event the role simultaneously manages players and

messages. Managing players consists in animating players except if a *pause* event occurs. Managing messages consists in looking for new messages dispatching a new strategy.

5.4 Protocol Description (PD)

The aim of this activity is to define purpose-specific interaction protocols whose need may arise when the description of communications done during the COD (Communication Ontology Description) and SD (Scenario Description) activities do not match any of the existing FIPA protocols.

The designer starts from scenarios and the ontological description of communications in order to find the need for a new specific interaction protocol. If this is the case, then he can proceed to the definition of a new protocol that is compliant with the interactions described in scenarios and the communication semantics.

It is advisable to refer to the FIPA Interaction protocols library[6] in order to see if a satisfying protocol already exists and if not, probably an existing one can be the basis for changes that can successfully solve the specific problem.

5.5 Organisation Dependencies Description (ODD)

In order to maximise its goal achievement expectation, an agent has to be able to estimate the competences of its future partners and to identify the most appropriate collaborators. The Capacity concept allows us to represent the competences of an agent or a set of agents. Capacity is a specialisation of the AbstractCapacity element. A Capacity describes what an Agent should be able to do in order to satisfy the requirements it is responsible for. This means that the set of Capacities obtained by refining the AbstractCapacity of the Problem Domain, becomes a part of the specification of the system requirements in the Agency Domain. Capacities describe what the agent can do at an abstract level, independently of how it does it (this is a concern that may be dealt with the Service concept). Capacity finds an implementation in the Service provided by the role and it is used to model what is done by an AgentTask in order to contribute in providing a service.

A service implements a capacity thus accomplishing a set of functionalities on behalf of its owner, a role (definition inspired from the Web Services Architecture [49] of the W3C[7]). These functionalities can be effectively implemented by a set of capacities required by the owner role. A role can thus publish several of its capacities and other members of the group can take profit of them by means of a service exchange. Similarly a group, that is able to provide a collective capacity can share it with other groups by providing a service.

The relationship between capacity and service is thus crucial in our metamodel. A capacity is an internal aspect of an organisation or an agent, while the service is

[6] FIPA Interaction Protocols specifications: http://www.fipa.org/repository/ips.php3
[7] World Wide Web Consortium: http://www.w3.org

designed to be shared among various organisation or entities. A service is created to publish a capacity and thus to allow other entities to benefit from it.

It will be assumed that an organisation is able to provide a service if the service is provided by of one of its roles. A role may provide a service if it manages/supervises a workflow where either:

- the service is implemented in one of its own behaviour, the role owns a capacity with a compatible description.
- the service is obtained from a subset of other roles of the same organisation by interacting with them using a known protocol; this is a service composition scenario where the work plan is a priori known and performance may be someway ensured,
- the service is obtained from the interaction of roles of the same organisation but the work plan is not a priori known and the service results in an emergent way.

In the last case, the management of a service provided by the global organisation behaviour is generally associated to one of the organisation's roles. It can also be managed by at least one of the representative of the various holons playing roles defined in the corresponding organisation.

Although capacities and services play a central role in this activity, the process to be performed does not start from them. Organisation Dependencies Description activity starts with the identification and description of resources that are manipulated by roles.

Resources in ASPECS are regarded as abstractions of environmental entities accessed by boundary roles; in order to access resources, roles need specific capacities that are now purposefully introduced (and then realised by services if necessary). In this way dependencies of organisations with the real world are made explicit.

Identifying resources often implies finding new capacities devoted to manipulate these resources. Moreover, since organisations depend on each other through service exchanges, services provided by roles (while exploiting their capacities and accessing resources) can now be identified. An AgentRole can be responsible for providing one of more services. A service description is provided by adopting one of the W3C specifications, namely RDF.

Finally, this activity should also outcome with the description of interfaces used by the system to manipulate resource. When capacities and services are fully described, a test is necessary to ensure that each capacity is associated with its set of possible service-level realisations. This matching between service and capacity allows the initialisation of a repository that may be used to inform agents on how to dynamically obtain a given capacity. Moreover it also proves that the system hierarchical decomposition is correct since the matching should validate the contribution that organisations acting at a given level give to upper-level organisations.

Figure 18 describes capacities, services and resources related to the three organisations involved in the model of the Robot Soccer Simulator case study. Contributions of each organisation to the upper level have been detailed by using a capacity and the associated service, e.g. *SimulateTeamBehavior* or *PlayStrategy*.

One new resource has been identified (it represents a logging file), and the *Log* capacity has been created to manage it. It is interesting to note that this capacity does not need a service realisation because the corresponding functionality is internal to the *GameObserver* role that does not need to publish it as a service.

This capacity does not represent a contribution between organisations, rather it is a competence required by a role to access one of its dependencies.

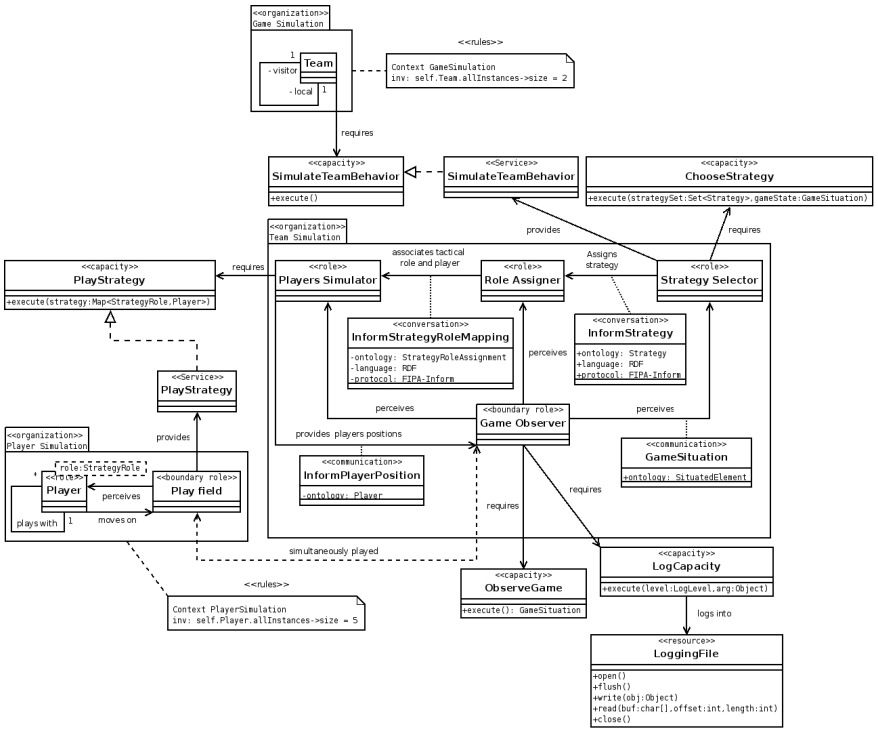

Fig. 18 Dependencies of the Team Simulation organisation

5.6 Role Constraints Identification (RCI)

This activity aims at identifying constraints between roles, such as roles that have to be played simultaneously, priorities, mutual exclusions, preconditions for one role generated by another, and so on. Concurrency constraints are also important because they will guide the definition of role scheduling policies. Detailed constraints between roles must prevent their inopportune concurrent execution and force the correct execution sequence. Roles shall be played simultaneously if and only if they allow an exchange of information between two different organisations. A means for realising this exchange can be the agent internal context when both roles belong to the same agent. This constitutes an alternative to the use of services and a simplification of information transfer.

Constraints between roles are identified thanks to roles dependencies and associated knowledge described in the previous activity. Role behaviour description also defines which information is eventually required from other organisations and it thus allows the identification of couples of roles that have to be played simultaneously.

In the presented case study, the environment is simulated by using one single agent. This implies that all boundary roles of the application have to be played by a single agent. This is described in figure 18, by a constraint between the two identified boundary roles (*Play Field* and *Game Observer*). This type of constraint will have relevant consequences on the structure of the final system as it will be detailed in section 5.7.

5.7 Holarchy Design (HD)

Two fundamental elements of the MAS metamodel are newly introduced in this activity; they are Agent and Holon. An Agent is an entity which can play a set of roles defined within various organisations; these roles interact each other in the specific context provided by the agent itself. Several AgentRoles are usually grouped in one Agent. The Agent context is given by its knowledge, its capacities and its environment. Roles share this context by the simple fact of being part of the same agent. This means for instance that an agent can play the role of *Buyer* in an organisation and later the same agent can sell the goods it had just acquired thus playing for the same organisation a different role (*Seller*); conversely, the same agent can also play roles belonging to another organisation (for instance devoted to monitor businesses), i.e. it can play the role (*AffairMonitor*) to trace the results and the performance exploited during the first acquisition process. It is worth to note that the agent is still not an implementation-level element (platform-dependent). Indeed, it needs further refinements. Figure 19(a) depicts the context defined by an agent as an interaction space for the roles it plays. These roles, in turn, belong to different organisations, each one defining a different context.

The concept of Holon is specialised from the Agent one. Two overlapping aspects have to be distinguished in the notion of Holon:

Administration and Holon Government: the first is directly related to the holonic nature of the entity (a holon, called super-holon, is composed of other holons, called sub-holons or members) and deals with the holonic structure itself is managed. This aspect is refered from now on as government. This aspect is common to every holon and thus called the *holonic* aspect. It describes the decision making process and how members organise and manage the super-holon.

Goal dependent Interactions: The second aspect is related to the problem to solve and the work to be done. It depends on the application domain and is called the *production* aspect. It describes action coordination mechanisms and interactions between members to achieve the objectives of the super-holons, the tasks to fulfil, or to take a decision.

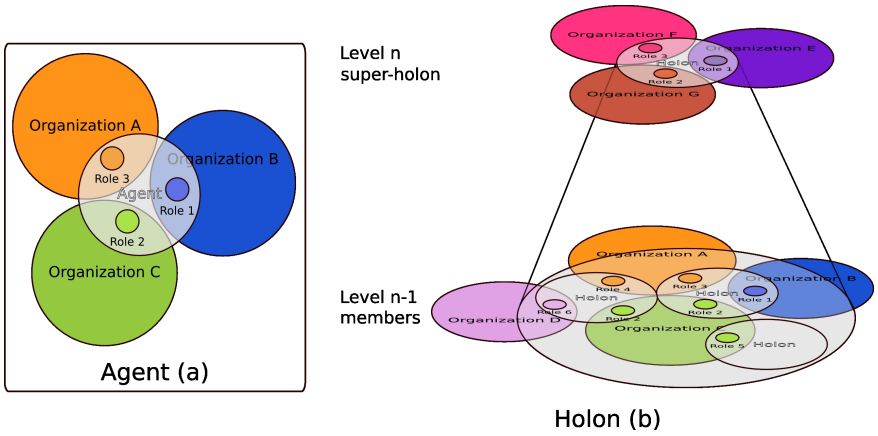

Fig. 19 Agent and Holon symbolic representation

Naturally our definition of holon integrates these two aspects and merges them within an organisational approach. A holon is thus a set of roles that can be defined on various organisations interacting in the specific context provided by the agent. A holon can play several roles in different organisations and it may be composed of other holons. A composed holon (super-holon) contains at least a single instance of a holonic organisation to precise how members organise and manage the super-holon and a set (at least one) of production organisations describing how members interact and coordinate their actions to fulfil the super-holon tasks and objectives. An atomic (non composed) holon is an AtomicAgent. Figure 19(b) illustrates this definition of holon.

In order to properly define the discussed aspects of each holon, the Holarchy Design activity is decomposed of four main tasks:

a. firstly, the agentification task consists in identifying all the required agents/holons and associating them with the set of roles they have to play.
   To achieve that, each previously identified organisation is specialised into a Group. This element is used to model groups of Agents that cooperate in order to achieve a goal. The association between Agents and AgentRoles allows the identification of the set of capacities that are required by AgentRole in order to be played by Agents.
b. The second task focuses on composed holons and aims at identifying a government type for each of them. The objective consists in describing the various rules used to take decisions inside each super-holon.
c. Then, all the previously described elements are merged in order to obtain the complete set of holons (composed or not) involved in the solution. In this way, the complete holarchy of the system is described.
d. The description obtained with the previous tasks is just the initial structure of the system, the last objective is now to specify holons' self-organisation mechanisms (creation, new member integration, scheduling policies for roles) in order to support a dynamic evolution of the system holarchy. Three main aspects

may be distinguished: Holon Member's modelling, Holon Government, Holon Dynamics. Each of them will be briefly detailed below.

Holon Member's modelling
This section deals with holonic intrinsic nature of the designed holons. The goal is to clarify how holon members are internally organised to generate and manage a super-holon. The *Holonic organisation* is defined to describe the government of a holon and its structure in terms of authority, power repartition, etc. This organisation is the basis of the holon government and it represents a *moderated group* (see [23]) in terms of roles (called *holonic roles*) and their interactions. In a moderated group, a subset of the members will represent all the sub-holons in the outside world. This management structure was adopted due the wide range of configurations it allows. Four roles are defined to describe the status of a member inside a super-holon:

Head, decision maker: it represents a privileged status conferring a certain level of authority.
Representative, interface of the holon: it is an externally visible part of a super-holon, it is an interface between the outside world (same level or upper level) and the other holon members. It may represent other members in taking decisions or accomplishing tasks (i.e. recruiting members, translating information, etc). The *Representative* role can be played by more than one member at the same time.
Part: Classical members. Normally in charge of doing tasks affected by head, a *Part* can also have an administrative duty, and it may be employed in the decision making process. It depends on the configuration chosen for modelling the super-holon. The *Part* role represents members belonging to only one super-holon.
Multi-Part: extension of *Part*. This role is played by sub-holons belonging to more than one super-holon.

The holonic organisation is instantiated into a holonic group. The Holonic-Group MAS metamodel element is a specialisation of group, and it is devoted to contain roles taking care of the holon internal decision-making process (composed holon's government).

Holon Government: Decision Making Process
In order to define a Holon Government one has to define a decision making process. For instance when an external holon is requesting its admission as a member, the decision to accept or refuse it should be taken according to a specific decision making mechanism that has to be defined (for instance, a voting mechanism may be used).

This section does not intend to present an extensive discussion about decision making procedures and patterns; the reader interested on the subject may refer to [50]. Trying to enumerate all possible issues would probably result in an incomplete or domain dependent list.

Two aspects of the decision making process should be analysed: (i) who is in charge of taking the decisions and how this happens (head, vote, etc); (ii) how the requesting process could be started (who is to be contacted by the external holon that wants to enter the super-holon).

The decision process for the admission of a new member can be done according to several different internal policies representing different levels of involvement of the holon members community: federation is positioned at one side of the spectrum, dictatorship on the opposite one. In the federation configuration, all members are equal when a decision has to be taken. Opposite to that, in dictatorship, heads are omnipotent; a decision taken by one of them does not have to be validated by any other member. In this government form, members lose most of their autonomy having to request permission of the head to provide a service or request a collective action.

Even if these configurations may be useful in specific domains, generally an intermediate configuration will be desirable. In order to identify the right choice, it is necessary to analyse the functionalities that will be provided by the holon, and then define the level of authority that the Head must have. A versatile solution to this problem consists in establishing a voting mechanism for each functionality.

Holon Dynamics

Because of space concerns, only the most common and important rules governing holon dynamics are discussed in this part, especially those dealing with members recruitment and holon creation. The first rule is about managing *Inclusion/Exclusion of holon's members*. Once a super-holon has been created, new members may request to join it or the super-holon itself may require new members to achieve its goal/task (the new member admission process is called **Merging**). In order to support the integration of new members, a "standard" interface should be provided to external holons to request their admission. A specific organisation with two roles, *StandAlone* (played by the candidate), and *Representative* (played by at least one of the representative of holons members), has been designed to manage this recruitment process. Another important aspect of the holon creation mechanism regards the motivations for the birth of a holon; these can in fact either depend on the need of satisfying in a collective way a requirement that cannot be accomplished by the single entities alone or on the need for improving the internal structure of an existing holon that is become too big and whose tasks are too complex to be managed. Concerning holon creation, it is therefore possible to distinguish two different mechanisms:

i) A top-down mechanism, the sub-division: a super-holon whose tasks are too complex, decides to create a set of internal organisations that are able to execute these tasks thus distributing the computational cost and breaking down organisational complexity. This case could be reduced to a specific case of the initial creation process, because newly created holons are configured to satisfy integration constraints with the super-holon.
ii) A bottom-up mechanism, the fusion (merging process): a set of holons decides to merge and to create a super-holon for satisfying a common goal. In this case, all rules that will govern the life of the super-holon have to be defined.

The final structure of the holonic solution dealing with our Robot Soccer simulator is presented in figure 20. At level 2 of the holarchy two super-holons (1 and 2) play the role Team in the group $g0$*: Game Simulation*. This denomination indicates that group $g0$ is an instance of the *Game Simulation* organisation. As such, members involved in the group, play one of the roles defined in the related organisation. Each of these two super-holons contains an instance of the *Team Simulation* organisation $g2$ and $g6$. Each composed holon (1, 2, 3, 7) contains a holonic group defining its governmental structure. Each *Team* holon disposes of a simple type of government inspired by the monarchy where command is centralised in the hands of a single head. The rule is that the holon playing the *Strategy Selector* role is automatically promoted Head and Representative of other members. The part holons 3 and 7 playing the *Players Simulator* role are decomposed, and each of them contains an instance of the *Player Simulation* organisation. Their government is inspired by the apanarchy[8] government type where each member is involved in the decision process (everybody is Head). The atomic holon 6, playing a MultiPart role, is shared in two couples of super-holons (1, 2) and (3, 7). This holon constitutes the environmental part of the application and plays all the boundary roles in agreement with the description made in the Role Constraints Identification phase.

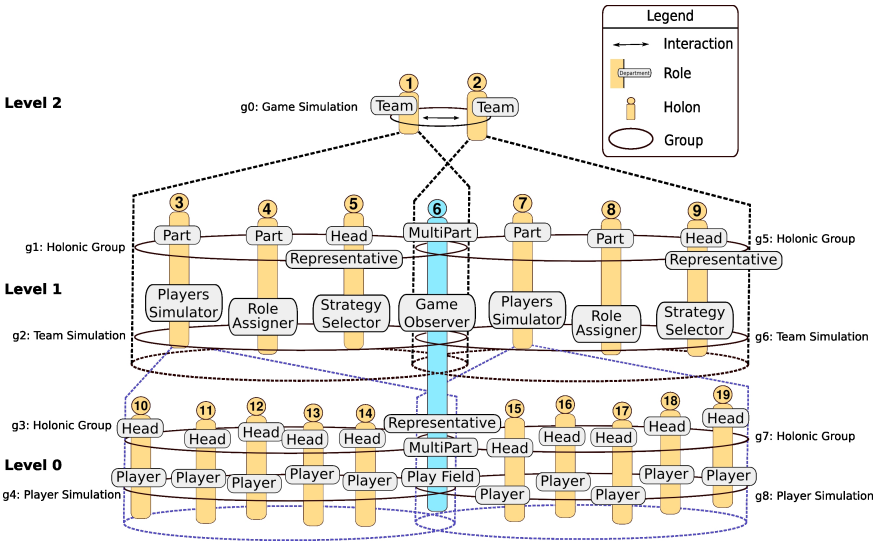

Fig. 20 The complete holonic structure of the agent-oriented part of the Robot Soccer Simulator

[8] The name is a composition of the Greek *Apan* meaning *all* or *every* and *archein*, "to rule".

6 Implementation

This section will give a brief overview of the implementation phase. The details of its activities and the set of associated diagrams are described in figure 21. Concepts at the basis of our implementation, and constituting the Solution Domain part of the HMAS meta-model are reported in Figure 22.

The Implementation phase aims at implementing the agent-oriented solution designed in the previous phase by adapting it to the chosen object-oriented implementation platform. This part of the work is thus dependent on the implementation and deployment platform. In this paper, *Janus*[9] is the chosen implementation platform.

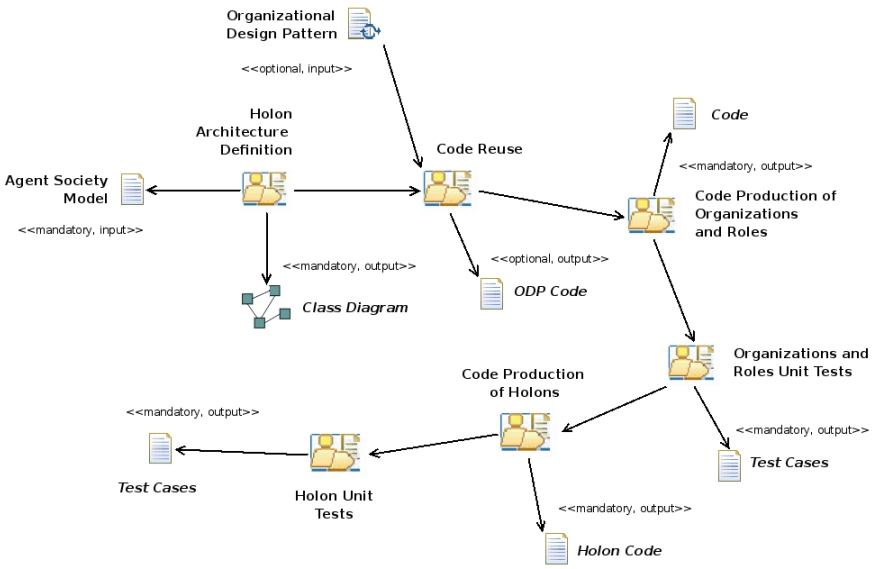

Fig. 21 Implementation Phase

The *Janus* platform was built in our lab for this purpose. It is specifically designed to deal with holonic and organisational aspects. The goal of *Janus* is to provide a full set of facilities for launching, displaying, developing and monitoring holons, roles and organisations.

The two main contributions of *Janus* are: (i) its native management of holons, and (ii) its implementation of the notion of *Role* as a concrete implementation-level entity. In contrast with other platforms such as MadKit [17], JADE, and FIPA-OS, in *Janus* the concept of Role is considered as a first class entity. It thus allows to directly implement organisational models without making any assumptions on the architecture of the holons that will play the role(s) of an organisation. An organisation is defined by a set of roles and a set of constraints to instantiate these roles

[9] http://www.janus-project.org

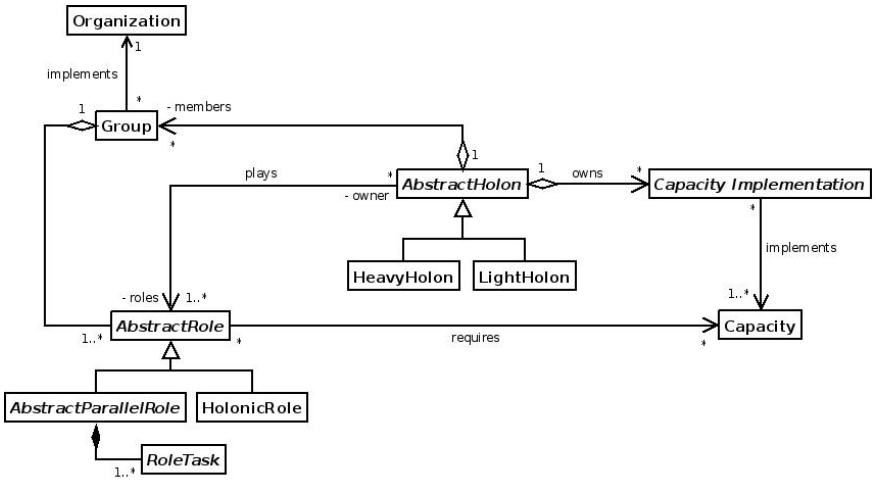

Fig. 22 The UML diagram of the Solution Domain of the ASPECS metamodel

(e.g. maximal number of authorised instances). Thus, organisations designed for an application can be easily reused for another. *Janus* so promotes reusability and modularity, moreover the use of organisational design patterns is strongly encouraged. Each organisation is a singleton and it can be instantiated by several groups. Group is the runtime context of interaction. It contains a set of roles and a set of Holons playing at least one of these roles. In addition to its characteristics and its personal knowledge, each agent/holon has mechanisms to manage the scheduling of its own roles. It can change dynamically its roles during the execution of the application (leave a role and request a new one). The life-cycle of a *Janus* agent is composed of three main phases: activation, life, termination. The life of an agent consists in consecutively executing its set of roles and capacities. To describe the personal competences of each agent/holon, *Janus* implements the concept of *JCapacity* that is an abstract description of a competence; each agent can be natively equipped with an implementation of a JCapacity or can dynamically acquire it (this function is still under development). In addition to the integration of these personal characteristics, a holon provides an execution context for roles and capacities.

Based on *Janus*, the implementation phase details activities that allow the description of the solution architecture and the production of associated source code and tests. Of course the process described in this phase can also be used with any other platform able to provide a translation of concepts presented in the solution domain of the meta-model.

6.1 Holon Architecture Definition

This activity aims at defining the architecture of each holon involved in the implementation of the previously design solution. Each organisation together with its set of roles and associated tasks have to be described. Each holon is associated with

the set of roles it should play, the set of capacities and services that it owns. Two different approaches may be used to design a holon. A static approach consists in designing a specific holon architecture for each architecture designed in the Holarchy Design activity. This approach is the most simple and easy to maintain, but it may generate a relevant number of different architectures in complex applications. Another approach consists in designing a dynamic holon architecture where holons will dynamically acquire roles and the corresponding set of required capacities. In this activity the designer also defines composed holons government rules and *JAtomicAgents* that may be used when an holonic structure is not required.

Figure 23 depicts a part of the Holons architecture implied in the implementation of our Robot Soccer Simulator example. In this example we have opted for a static approach. For each architecture identified in the Holarchy Design activity, a specific holon has been designed. Three holon architectures and three organisations are partially described in the class diagram. Only the role *JPlayersSimulator* is described in terms of *JRoleTask*. Classes already introduced in the solution domain (see figure 22) correspond to classes provided by the *Janus* platform that are refined to introduce problem dependent artefacts.

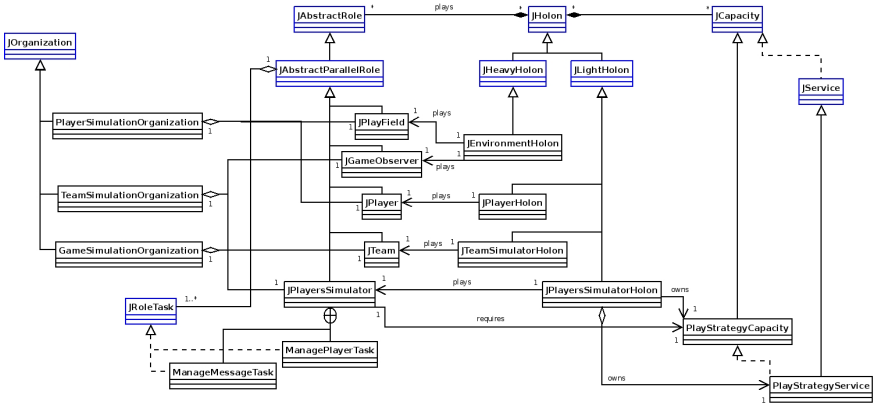

Fig. 23 The Holon Architecture designed for the proposed case study

6.2 Code Reuse

A set of organisational patterns may have been used during the two previous phases especially for the OID (Organisation Identification) and IRI (Interactions and Roles Identification) activities and for the Holon Government identification task. This activity aims at integrating the code of these patterns inside the currently designed application. It also intends to provide a framework for reusing code of previous applications that can be reused in the current one. Integrating patterns source code may require some adaptation work; for instance it is often necessary to adapt the managing of input values provided by the remaining part of the application to the reused code.

6.3 Code Production of Organisations and Roles

This activity aims at producing the code for organisations and roles. They are the most elementary building blocks of the *Janus* platform; actually each role and organisation becomes a class in the code and they are grouped in specific packages (one for each organisation). Starting from the structural and dynamical representation of roles and organisations the programmer can code their implementation using the *Janus* primitives. It is part of our future works to provide tools for the automatic generation of these portions of code from design diagrams.

6.4 Organisations and Roles Unit Tests

The approach used for Test in ASPECS consists in successively testing each context from the role (the smaller one) to the entire system. This activity aims at testing behaviours that will be used to compose the system; this means individual behaviours represented by roles and global behaviours corresponding to organisations.

Each role is individually tested; stimuli coming from other roles defined within the same organisation and stimuli coming from the agent context through capacities are emulated (here the referred agent is the one that should play the role under test). This kind of test involves the definition of stubs and drivers like it happens in conventional object-oriented unit test. Each organisation is then instantiated in a runtime context called group. Its roles are instantiated too. This task consist in validating roles interaction scenarios inside an organisation, stimulus coming from agents contexts are emulated. From the inputs artefacts, it is possible to produce a set of test cases that can verify the obedience of the obtained roles and organisations to the prescribed specifications. After writing test cases and coding necessary support elements, tests can be performed and identified failures will lead to new iterations to fix related bugs.

6.5 Code Production of Holons

This activity focuses on code production for holons. In the *Janus* platform, each holon is represented by a class. *Janus* offers two main kinds of holon: a threaded and a non-threaded one. The programmer has to choose the most appropriate one for the specific problem.

Starting from the results of the Holarchy Design activity, the programmer chooses the most adapted version of *JHolon* and can code the holon implementation by using the associated *Janus* primitives. When a non-threaded implementation is chosen, holon scheduling aspects have to be coded too. The three methods that govern the life-cycle of each holon have also to be defined (*activate(), live(), end()*); they are associated to the three main states of the holon's life: activation, execution, and termination. As inspired by the Madkit synchronous engine[10], *Janus* provides a full set of tools to manage non-threaded holons execution.

[10] refer to http://www.madkit.net/site/madkit/doc/devguide/synchronous.html

6.6 Holon Unit Tests

Holon Unit test is the second level of test in the ASPECS process. It aims at validating the global holon's behaviour. A particular attention is paid to holon dynamics especially to rules governing holon creation, management (task attribution) and the process of members integration. Each holon is individually tested. External stimuli are again emulated by external stub/drivers. After writing test cases and coding necessary support elements, tests can be performed and identified failures will lead to perform new iterations where bugs will be fixed.

7 Deployment

The Deployment phase is the final one in the ASPECS software development process. Figure 24 describes the two activities of this phase and their associated work products, further details will be provided in the two following sub-sections.

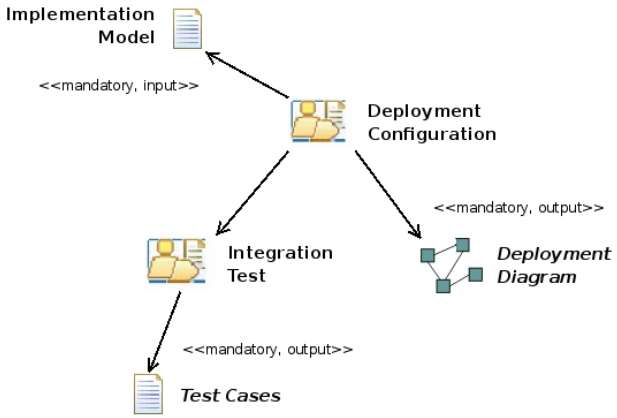

Fig. 24 Deployment Phase

This phase aims at detailing how to deploy an application over various *Janus* kernels. *Janus* adopts a peer to peer technology (based on JXTA[11]) for allowing kernels federation and agents migration.

7.1 Deployment Configuration

This activity aims at detailing how the previously developed application will be concretely deployed; this includes studying distribution aspects, holons physical location(s) and their relationships with external devices (sensors, actuators used/accessed by agents) and resources. This activity also details how to perform

[11] JXTA, Juxtapose: https://jxta.dev.java.net

the integration of parts of the application that have been designed and developed with traditional approaches (i.e. object-oriented ones) with parts designed by using an agent-oriented approach.

The first task of this activity consists in establishing a partitioning among the various holons used to develop the application. This partition is mainly performed according to resources localisation and the organisation in which they are used. Then at least one *Janus* kernel is deployed on each available elaborating unit. At this stage, the set of corresponding holons and their associated organisations are deployed on the various kernels according to the previously defined partition. If a dynamic discovery process is used to integrate at runtime new *Janus* kernels, the way to deploy organisations and their associated roles on newly discovered kernels has also to be described.

Figure 25 partially describes the deployment configuration associated to our Robot Soccer simulator example. Three *Janus* kernels are used to distribute the application over the network: one kernel is used for each of the two JTeamSimulatorHolon holons (and their associated members) and the last one is used for the shared environment holon and the graphical user interface. In our example we assume that all kernels already have installed all the classes associated to each organisation involved in the designed solution.

7.2 Integration Test

Integration test is the final activity of the ASPECS process. This activity aims at verifying if the system effectively fulfils all the requirements identified in the Domain Requirement Description activity (a great relevance is now given to verification of non functional requirements that hardly can be tested in the previous test activities). The purpose of integration testing is also to detect any inconsistency between the software units that are integrated together and among these units and the hardware on which they are deployed.

Individual software modules like organisations and holons are tested as a single application according to the defined test cases.

8 Conclusion and perspectives

This paper presented the ASPECS software development process. This process has been entirely detailed and illustrated with the help of a concrete case study from the requirement analysis activities to deployment of the system on a specific platform developed in our lab. ASPECS covers the entire software engineering process and it is designed for the development of complex software systems, especially (but not exclusively) those exhibiting a hierarchical structure. A set of methodological guidelines has been provided for the majority of the activities.

The respect and integration of the most diffused AOSE domain standard specifications are one of the bases of our approach. The description of the development process is thus based on SPEM, graphical notations are based on UML and FIPA

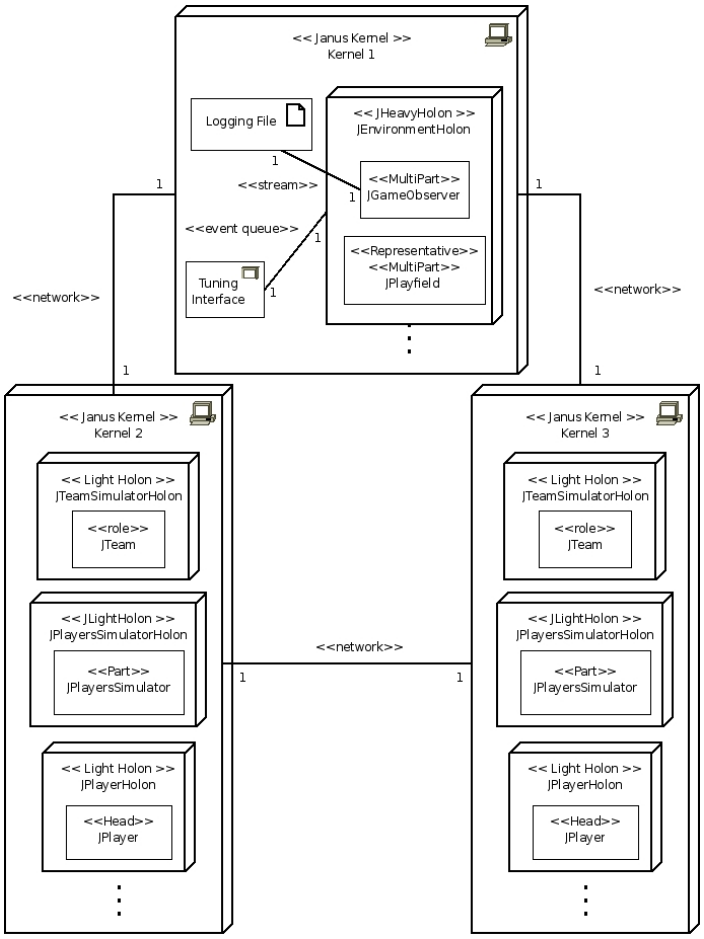

Fig. 25 Deployment Diagram of the Robot Soccer Simulator

standards are also largely adopted. ASPECS notations extend UML especially to take into account the organisation and holon concepts.

ASPECS allows the modelling of a system at an arbitrary number of abstraction levels through a hierarchical behavioural decomposition based on roles and organisations. The system is recursively decomposed down to a level where behaviours are considered as sufficiently simple to be manageable by atomic easy-to-implement entities. Contributions between two adjacent levels of abstraction are modelled thanks to the relation between the concepts of capacity and service.

Thanks to the introduction of the notion of capacity, organisations and their associated roles may be defined without making any assumptions on entities architecture. This enables the definition of generic organisation models that facilitates design reusability and extensions.

Concerning the environment that is an essential part of MAS, ASPECS through the use of boundary roles explicits the representation of interactions between the

system and the necessary environmental entities without making any assumptions on the concrete environment structure. The use of specific capacities as an the interface between the environment and the system renders easy the deployment of applications on dynamic and heterogeneous environments.

Domain knowledge in the system is explicitly encoded in the Problem and Solution ontologies. ASPECS thus presents an holistic model of the structure of the domain knowledge and the interaction and dependencies of knowledge components in the system. This approach allows an easy sharing, reusability, extension and maintainability of system knowledge in a modular manner.

ASPECS is part of a larger effort aiming at providing a complete methodology with the associated set of notations and tools to support design activities from requirement analysis to code generation and deployment. Two major tools are currently under development in our lab. The first is the *Janus* platform that is already well advanced. *Janus* is dedicated to implementation and deployment of holonic applications. And the second is Janeiro, a CASE tools that deals with the analysis and design aspects.

Associated to the *Janus* platform, ASPECS allows an easy implementation of models while maintaining the advantages of the organisational approach. At the entity level it enables a dynamic reasoning and extension of agents/holons capabilities at runtime (through the implementation of capacity).

The chosen case study confirms that the holonic organisational approach is able to deal with complex software development and proves the scalability and modularity of the proposed approach.

Further works will particularly focus on the integration of formal notations and methods especially OZS. OZS (Object-Z and Statechart [26]) has been already used for role behaviour description where roles are formally described by using an Object-Z class. In these cases, the behaviour of the role is described using a statechart where associated methods refer to formal defined ones. Procedures to automatically generate templates of role code from OZS behavioural specifications are also under development (they will implement an automatic translation of statecharts to Java code).

A complete integration of Object-Z in the Domain Requirement Engineering phase is also planned. The objective consists in providing a formal description to the activities that could take an advantage from that. This aspect will increase the solution quality and also facilitate the automatic generation of test cases.

Acknowledgements Authors would like to thanks Sebastiàn Rodriguez for having substantially contributed to this work by proposing a framework for the conception of Holonic multiagent systems that is at the basis of the ASPECS metamodel.

References

1. F. Bergenti, M.-P. Gleizes, and F. Zambonelli, editors. *Methodologies and Software Engineering for Agent Sytems*. Klüwer Academic Press, 2004.

2. F. Bergenti and A. Poggi. Exploiting uml in the design of multi-agent systems. In A. Omicini, R. Tolksdorf, and F. Zambonelli, editors, *Engineering Societies in the Agents' World*, LNAI. Springer Verlag, 2000.
3. C. Bernon, V. Camps, M.-P. Gleizes, and G. Picard. Engineering adaptive multi-agent systems: The adelfe methodology. In B. Henderson-Sellers and P. Giorgini, editors, *Agent-Oriented Methodologies*, volume ISBN1-59140-581-5, pages 172–202. Idea Group Publishing, NY, USA, juin 2005.
4. C. Bernon, M. Cossentino, and J. Pavón. An overview of current trends in european aose research. *Informatica*, 29(4):379–390, July 2005.
5. P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, and A. Perini. Tropos: An agent-oriented software development methodology. *Journal of Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, 2004.
6. H. V. Brussel, J. Wyns, P. Valckenaers, L. Bongaerts, and P. Peeters. Reference architecture for holonic manufacturing systems: Prosa. *Computers in Industry*, 37:255–274, 1998.
7. G. Caire, W. Coulier, F. J. Garijo, J. Gomez, J. Pavón, F. Leal, P. Chainho, P. E. Kearney, J. Stark, R. Evans, and P. Massonet. Agent oriented analysis using message/uml. In M. Wooldridge, G. Weiß, and P. Ciancarini, editors, *Agent-Oriented Software Engineering II, Second International Workshop, AOSE 2001, Montreal, Canada*, volume 2222 of *LNCS*, pages 119–135. Springer Verlag, 2002.
8. A. Chella, M. Cossentino, L. Sabatucci, and V. Seidita. Agile passi: An agile process for designing agents. *International Journal of Computer Systems Science & Engineering. Special issue on Software Engineering for Multi-Agent Systems*, 21(2), March 2006.
9. M. Cossentino. From requirements to code with the passi methodology. In B. Henderson-Sellers and P. Giorgini, editors, *Agent-Oriented Methodologies*, chapter IV, pages 79–106. Idea Group Publishing, Hershey, PA, USA, 2005.
10. M. Cossentino, S. Gaglio, A. Garro, and V. Seidita. Method fragments for agent design methodologies: from standardization to research. *International Journal on Agent Oriented Software Engineering (IJAOSE)*, 1(1):91–121, April 2007.
11. M. Cossentino, N. Gaud, S. Galland, V. Hilaire, and A. Koukam. A holonic metamodel for agent-oriented analysis and design. In V. Marik, V. Vyatkin, and A. Colombo, editors, *Proc. of the 3rd International Conference on Industrial Applications of Holonic and Multi-Agent Systems (HoloMAS'07)*, volume 4659 of *Springer-Verlag LNAI*, pages 237–246, Regensburg, Germany, September 2007.
12. M. Cossentino and V. Seidita. Composition of a new process to meet agile needs using method engineering, vol. iii. In *Software Engineering for Large Multi-Agent Systems*, volume 3390 of *LNCS*, pages 36–51. Springer-Verlag GmbH, 2005.
13. S. DeLoach. Multiagent systems engineering: a methodology and language for designing agent systems. In *Agent Oriented Information Systems '99*, 1999.

14. A. Drogoul and J. Zucker. Methodological issues for designing multi-agent systems with machine learning techniques: Capitalizing experiences from the robocup challenge, 1998.
15. M. Edwards. A brief history of holons, October 2003. available at http://www.integralworld.net/edwards13x.html.
16. J. Ferber and O. Gutknecht. A meta-model for the analysis and design of organizations in multi-agent systems. In *ICMAS'98*, pages 128–135, 1998.
17. J. Ferber, O. Gutknecht, and F. Michel. From agents to organizations: an organizational view of multi-agent systems. In *Agent-Oriented Software Engineering IV 4th International Workshop*, volume 2935 of *LNCS*, pages 214–230, Melbourne, Australia, mar 2004. Springer Verlag.
18. Foundation For Intelligent Physical Agents. *FIPA RDF Content Language Specification*, 2001. Experimental, XC00011B.
19. Foundation For Intelligent Physical Agents. *FIPA ACL Message Structure Specification*, 2002. Standard, SC00061G.
20. Foundation For Intelligent Physical Agents. *FIPA Communicative Act Library Specification*, 2002. Standard, SC00037J.
21. S. Galland, F. Grimaud, P. Beaune, and J.-P. Campagne. Simulation of distributed industrial systems - a multi-agent methodological approach. In A. Dolgui, J. Soldek, and O. Zaikin, editors, *Supply Chain Optimisation: Product/Process Design, Facility Location and Flow Control*, volume 94 of *Applied Optimization*. Springer, 2005.
22. L. Gasser. Mas infrastructure: Definitions, needs and prospects. In *Revised Papers from the International Workshop on Infrastructure for Multi-Agent Systems*, pages 1–11, London, UK, 2001. Springer-Verlag.
23. C. Gerber, J. Siekmann, and G. Vierke. Holonic multi-agent systems. Technical Report DFKI-RR-99-03, DFKI - GmbH, 1999.
24. A. Giret and V. Botti. From system requirements to holonic manufacturing analysis. *International Journal of Production Research*, 44(18-19):3917–3928, 2006.
25. P. Gruer, V. Hilaire, A. Koukam, and K. Cetnarowicz. A formal framework for multi-agent systems analysis and design. *Expert Systems with Applications*, 23, December 2002.
26. P. Gruer, V. Hilaire, A. Koukam, and P. Rovarini. Heterogeneous formal specification based on object-z and statecharts: semantics and verification. *Journal of Systems and Software*, 70(1-2):95–105, 2004.
27. B. Henderson-Sellers. Method engineering for oo systems development. *Commun. ACM*, 46(10):73–78, 2003.
28. V. Hilaire, A. Koukam, P. Gruer, and J.-P. Müller. Formal specification and prototyping of multi-agent systems. In A. Omicini, R. Tolksdorf, and F. Zambonelli, editors, *Engineering Societies in the Agents' World*, number 1972 in LNAI. Springer Verlag, 2000.
29. N. Jennings. On agent-based software engineering. *Artificial Intelligence*, 177(2):277–296, 2000.
30. N. Jennings. An agent-based approach for building complex software systems. *Commun, ACM*, 44(4):35–41, April 2001.

31. E. A. Kendall. Role modeling for agent system analysis, design, and implementation. *IEEE Concurrency*, 8(2):34–41, 2000.
32. E. A. Kendall, P. V. M. Krishna, C. B. Suresh, and C. G. V. Pathak. An application framework for intelligent and mobile agents. *ACM Computing Surveys*, 32(1), 2000.
33. A. Koestler. *The Ghost in the Machine*. Hutchinson, 1967.
34. B. Kristensen and K. Osterbye. Roles: Conceptual abstraction theory and practical language issues. *Theory and Practice of Object Systems*, 2(3):143–160, 1996.
35. P. Kruchten. *The Rational Unified Process: An Introduction*. Addison-Wesley Professional, 3rd edition, 2004.
36. F. Maturana. *MetaMorph: an adaptive multi-agent architecture for advanced manufacturing systems*. PhD thesis, The University of Calgary, 1997.
37. J. Odell, M. Nodine, and R. Levy. A metamodel for agents, roles, and groups. In J. Odell, P. Giorgini, and J. Müller, editors, *Agent-Oriented Software Engineering (AOSE) IV*, LNCS. Springer, 2005.
38. J. Odell, H. Parunak, and B. Bauer. Extending uml for agents. In E. Y. G. Wagner and Y. Lesperance, editors, *Information Systems Workshop at the 17th National conference on Artificial Intelligence*, pages 3–17, 2000.
39. A. Omicini. SODA: Societies and infrastructures in the analysis and design of agent-based systems. In Springer-Verlag, editor, *Agent-Oriented Software Engineering: First International Workshop, AOSE*, volume 1957 of *Lecture Notes in Computer Science*, pages 185–193, 2000.
40. J. Pavón, J. Gómez-Sanz, and R. Fuentes. The ingenias methodology and tools. In B. Henderson-Sellers and P. Giorgini, editors, *Agent-Oriented Methodologies*, volume ISBN1-59140-581-5, pages 236–276. Idea Group Publishing, NY, USA, juin 2005.
41. S. Rodriguez, N. Gaud, V. Hilaire, S. Galland, and A. Koukam. An analysis and design concept for self-organization in holonic multi-agent systems. In S. Bruckner, S. Hassas, M. Jelasity, and D. Yamins, editors, *Engineering Self-Organising Systems*, volume 4335 of *LNAI*, pages 15–27. Springer-Verlag, 2007.
42. S. Rodriguez, V. Hilaire, and A. Koukam. Fomal specification of holonic multi-agent system framework. In *Intelligent Agents in Computing Systems, ICCS(3)*, number 3516 in LNCS, pages 719–726, 2005.
43. C. Rolland and N. Prakash. A proposal for context-specific method engineering. In *Proc. of the IFIP TC8, Working conference on method engineering on Method engineering : principles of method construction and tool support*, pages 191–208. Chapman & Hall, Ltd., 1996.
44. J. Searle. *Speech Acts*. Cambridge University Press, Cambridge, UK, 1969.
45. W. Shen, F. Maturana, and D. H. Norrie. Metamorph ii: an agent-based architecture for distributed intelligent design and manufacturing. *Journal of Intelligent Manufacturing*, 11(3):237–251, June 2000.
46. H. A. Simon. *The Science of Artificial*. MIT Press, Cambridge, Massachusetts, 3rd edition, 1996.

47. I. Sommerville. *Software Engineering*. International Computer Science Series. Addison Wesley, Pearson Education, seventh edition edition, 2004.
48. SPEM. *Software Process Engineering Metamodel Specification, v2.0, Final Adopted Specification, ptc/07-03-03*. Object Management Group (OMG), March 2007.
49. W3C. *Web Services Architecture*. World Wide Web Consortium, February 2004.
50. G. Weiss, editor. *Multiagent systems: a modern approach to distributed artificial intelligence*. MIT Press, Cambridge, MA, USA, 1999.
51. K. Wilber. *Sex, Ecology, Spirituality*. Shambhala, 1995.
52. F. Zambonelli, N. Jennings, and M. Wooldridge. Developing multiagent systems: the gaia methodology. *ACM Trans. on Software Engineering and Methodology*, 12(3), 2003.