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 / 1^{st} Revision: Apr. 25, 2008 / 1^{st} Revised version: June 15, 2008 / 2^{nd} Revision: Feb. 11, 2009 / 2^{nd} Revised version: May 21, 2009

Abstract Holonic multiagent systems (HMAS) offer a promising software engineering approach for developing complex open software systems. However the process of building Multi-Agent Systems (MAS) and HMAS is mostly different from the process of building more traditional software systems as 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 set 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 design process and the 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 by introducing new specific profiles.

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 [35, 54]. However, to deal with all aspects of complex systems MAS must deal with multiple levels of abstractions and openness, which is not the case for most solutions [41].

According to Simon [48], complex systems often (if not always) exhibit a hierarchical configuration[1]. The idea is that the architecture of a complex system can be explained and understood using hierarchical organisation structures as presented in [52]. Several metamodels and methodologies have been proposed for MAS [3]. However, most of them consider agents as atomic entities. There is no intuitive or natural way to deal with hierarchical organisation structures. Considering agents as composed entities thus enables the modelling of nested hierarchies and proposes a solution to this problem.

In the reported landscape, this paper advocates the use of holonic multiagent systems (HMAS) in which holons are agents that may be composed of agents for developing complex software systems. It introduces an agent-oriented software process for engineering complex systems called ASPECS. The process can be considered as an evolution of the PASSI [13] process for modelling HMAS and it also collects experiences about holon design coming from the RIO approach [33]. The construction of the new process has been performed according to the situational method engineering paradigm [32] and the approach described in [14]. 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 built according to this guideline [14]. This metamodel defines 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 complex system until the requested tasks are manageable by atomic easy-to-implement entities. In multiagent systems, 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.

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

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* [37]). 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 instance PROSA [6] and MetaMorph [47]. 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. Several methodologies and metamodels with a clear organisational vision have been already proposed, like: AGR [20], RIO [33], GAIA [54], INGENIAS [44], MESSAGE [7], and SODA [42]. 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 [54]. As pointed out by Ferber [20], the 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, and security of applications.

The objective of the proposed work consists in trying to gather the advantages of organisational approaches as well as those of the holonic vision in modelling complex systems. The result is a set of organisation-oriented abstractions that have been integrated into a complete methodological process called ASPECS.

In this paper, the ASPECS design process is illustrated by using a case study lying in the area of Holonic Industrial Systems. This case study aims at helping the manager of one of the most important automotive manufacturing plant in eastern France to evaluate different configurations for the production plant. Each configuration consists in a different allocation of manufacturing tasks to plant buildings. This case study will be introduced in Subsection 2.3 after the description of the ASPECS metamodel. In the rest of this paper, we refer to this case study by labelling it AMP (Automotive Manufacturing Plant).

The paper is organised as follows. Section 2 provides an early glimpse of the ASPECS process and modelling approach. It also introduces the metamodel and the core concepts of the methodology. Each phase of the ASPECS software process and their associated activities are then described in Sections 3–5, while they are applied to the AMP case study. An evaluation and comparison between ASPECS and nine agent-oriented software engineering methodologies is presented in section 6. Finally, section 7 summarises the results of the paper and describes some future work directions.

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 [13] and RIO [33] 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 of holonic (as well as not-holonic) multiagent systems. The idea underpinning the ASPECS design process can be described in a few fundamental choices that characterise the vision of the authors and represent some of the main scientific contributions of this work:

1. The ASPECS design process explicitly deals with the design of open, dynamic and complex systems. The main limit in its application scope is that we assume the system can be hierarchically decomposed in sub-systems (nearly-decomposable systems [48]).
2. The adoption of an organisational approach. Functionalities to be realised are assigned to organisations that accomplish them also by means of the hierarchical decomposition of the organisation structure in sub-organisations (holonic paradigm). An organisation-oriented analysis is applied by using two different decomposition strategies: vertical and horizontal. Vertical decomposition allows to delegate the responsibility of an organisation at level n to sub-organisations at level n-1 (lower abstraction level, finer grained organisations). Horizontal decomposition allows the collaboration of several entities at the same level of abstraction in order to fulfil the required functionalities.
3. Domain related ontological knowledge is used as a tool for enhancing the quality of design. This has been already adopted in some previous methodologies [34] but it is lacking in most modern approaches. We think that in dealing with intelligent agents it is particularly important to explicitly catch an ontological model of the problem and solution domains; this allows an easy application of several AI techniques as well as the adoption of semantic-based communications among agents.
4. We considered three main levels of abstractions in design, called models according to the model-driven engineering terminology. They are inherited from PASSI metamodel domains and they are: problem, agency and solution. Concepts of the problem domain are used to model system requirements in terms of organisations and interacting roles; concepts of the agency domain are the result of a set of transformations from the previous domain and are used to depict an agent-oriented solution; concepts of the solution domain are again the result of some transformations and are devoted to design a platform-specific solution at the code level.
5. The joint use of holonic and agency concepts to respectively model the two different faces of an entity that composes the system. Holonic concepts focus on the modelling of collective and compositional aspects of the system. While agent-related concepts focus on modelling individual aspects and personal goals of the entities composing the system.

Further details about these choices and their consequences on the resulting process will be provided throughout the paper. In the following subsections we will briefly introduce the ASPECS process structure, the ASPECS metamodel with the defini-

tions of the most important elements composing it and, finally, the case study that will be used throughout the paper.

2.1 ASPECS*: the process*

The conception of the ASPECS process has been based on concepts coming from Situational Method Engineering [32] and the work done by some of the authors in applying this paradigm to agent-oriented design methodologies [14]. The resulting ASPECS process also benefits from experiments and theoretical studies done on PASSI and Agile PASSI [9, 13]. Just like them, ASPECS is based on an iterative-incremental life-cycle, as it is for other widely accepted approaches in both the agent- [4, 38, 43, 44] and object-oriented contexts.

As regards its deliverables, 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 especially by the introduction of new profiles. In fact UML diagrams are often used to represent concepts that are not completely considered in UML and the notation has been modified to better fulfil the need of modelling agents.

The ASPECS process structure is based on the Software Process Engineering Metamodel (SPEM) specification proposed by OMG [50]. This specification is based on 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*. According to this metamodel, the ASPECS process is based on three main granularity levels of process components: *Phases*, *Activities* and *Tasks*. A *Phase* delivers a composite work product, composed by one or more documents that can belong to different work product types; it is composed of a number of *activities* that are in turn decomposable into tasks. An *Activity* delivers a main work product such as 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.

The description of the ASPECS software development process has been split in two different levels: in the first one (discussed in this section) we describe the process at the phase level; in the second one we report the details of each phase, with the associated activities, in a separate section (see sections 3-5). Because of space concerns, each activity has been only briefly described but the interested reader may refer to the ASPECS website[2] for a more complete description including another case study.

The ASPECS life cycle consists of three phases that are briefly described below. Phases are also depicted in Figure 1 where each activity is reported together with the main goal(s) it pursues.

The **System Requirements** phase aims at identifying a hierarchy of organisations, whose global behaviour may fulfil the system requirements under the chosen

[2] ASPECS: http://www.aspecs.org/

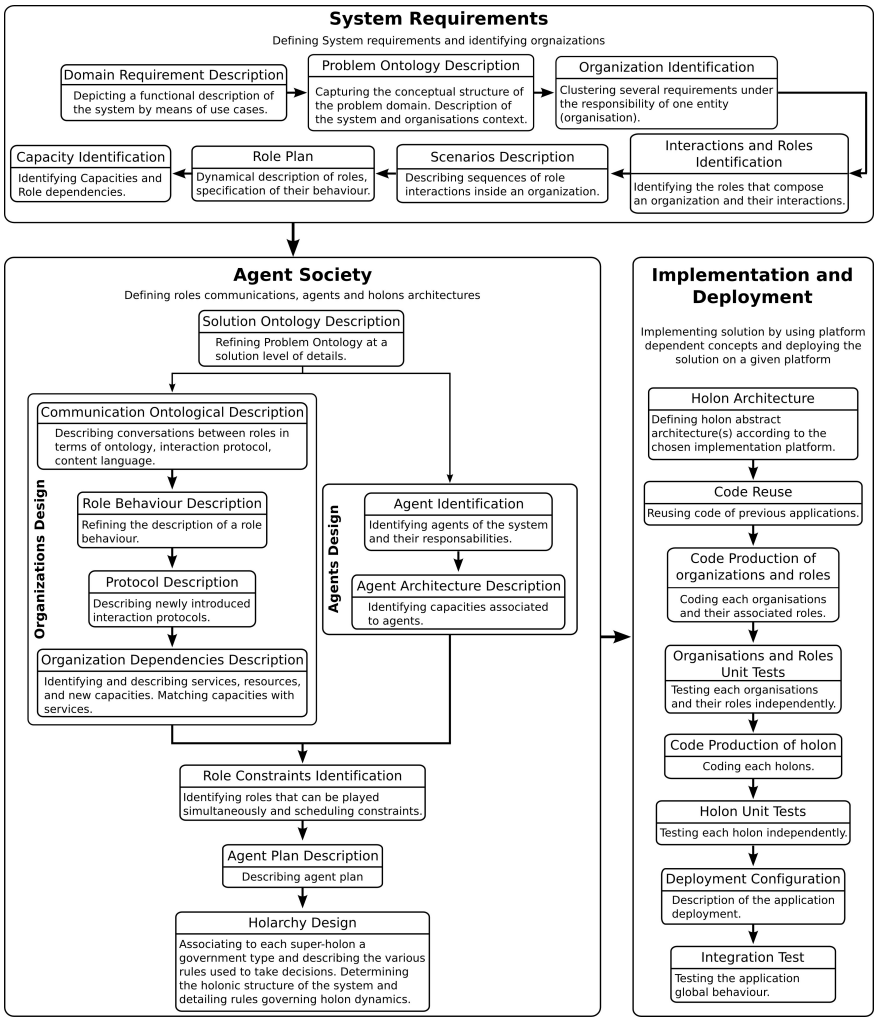

Fig. 1 Roadmap of the ASPECS process (Phases/Activities and their goals)

perspective. It starts with a Domain Requirements Description activity where requirements are identified by using classical techniques such as use cases. Domain knowledge and vocabulary associated to the problem domain are then collected and explicitly described in the Problem Ontology Description activity. Then, requirements are associated to newly defined organisations. Each organisation will therefore be responsible for exhibiting a behaviour that fulfils the requirements it is responsible for. This activity is called Organisation Identification and it produces an initial hierarchy of organisations that will later be extended and updated, with further iterations, in order to obtain the global organisation hierarchy representing the system structure and behaviour. The behaviour of each organisation is realised by a set of interacting roles whose goals consist in contributing to the fulfilment

of (a part of) the requirements of the organisation within which they are defined. In order to design modular and reusable organisation models, roles are specified without making any assumptions on the structure of the agent that may play them. To meet this objective, the concept of capacity has been introduced. A capacity is an abstract description of a know-how, i.e. a competence of a role. Each role requires certain skills to define its behaviour and these skills are modelled by means of a capacity. Besides, an entity that wants to play a role has to be able to provide a concrete realisation for all the capacities required by the role. Finally, the last step of the system requirements phase: the capacity identification activity, aims at determining the capacities required by each role.

The second phase is the **Agent Society Design** phase that 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. The objective is to provide a model in terms of social interactions and dependencies among entities (holons and agents). Previously identified elements such as ontology, roles and interactions, are now refined from the social point of view (interactions, dependencies, constraints, etc). At the end of this design phase, the hierarchical organisation structure is mapped into a holarchy (hierarchy of holons) in charge of realising the expected behaviours. Each of the previously identified organisations is instantiated in form of groups. Corresponding roles are then associated to holons or agents. This last activity also aims at describing the various rules that govern the decision-making process performed inside composed holons as well as the holons' dynamics in the system (creation of a new holon, recruitment of members, etc). All of these elements are finally merged to obtain the complete set of holons involved in the solution.

The third and last phase, namely **Implementation and Deployment** firstly aims at implementing the agent-oriented solution designed in the previous phase by deploying it to the chosen implementation platform, in our case, *Janus*. Secondly, it aims at detailing how to deploy the application over various computational nodes (*Janus* kernels in our experiments). 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. It also deals with the solution reusability by encouraging the adoption of patterns. The code reuse activity aims at integrating the code of these patterns and adapting the source code of previous applications inside the new one. It is worth to note that although we will refer to a *Janus*-based implementation, system developed by using other platforms can be designed as well with the described process. This phase ends with the description of the deployment configuration; it also details how the previously developed application will be concretely deployed; this includes studying distribution aspects, holons physical location(s) and their relationships with external devices and resources. This activity also describes how to perform the integration of parts of the application that have been designed and developed by using other modelling approaches (i.e. object-oriented ones) with parts designed with ASPECS.

2.2 ASPECS*: the metamodel and key concepts*

ASPECS has been built by adopting the Model Driven Architecture (MDA) [40] and thus we defined three levels of models each referring to a different metamodel. We also label the three metamodels "domains" thus maintaining the link with the PASSI metamodel that was one of our inspiration sources. The three domains we define are:

The **Problem Domain**. It provides the organisational description of the problem independently of a specific solution. The concepts introduced in this domain are mainly used during the analysis phase and at the beginning of the design phase (see Figure 2).

The **Agency Domain**. It introduces agent-related concepts and provides a description of the holonic, multiagent solution resulting from a refinement of the Problem Domain elements (see Figure 3).

The **Solution Domain** is related to the implementation of the solution on a specific platform. This domain is thus dependent on a particular implementation and deployment platform (see Figure 5). In our case, this part of the process is based on the *Janus* platform that we specifically designed to ease the implementation of holonic and organisational models. A complete description of the *Janus* platform would take too much space to be dealt by this paper and therefore we prefer to present only the most significant *Janus* issues, the interested reader can find more details in [24] and on the *Janus* website[3].

The following sub-sections detail the three domain metamodels and fundamental concepts within them. A complete description of all the elements reported in the metamodels is present on the ASPECS website and will not be reported here.

2.2.1 Problem Domain

The Problem Domain metamodel (see Figure 2) includes elements that are used to catch the problem requirements and perform their initial analysis: Requirements (both functional and non-functional) are related to the organisation that fulfils them. An organisation is composed of Roles, which are interacting within scenarios while executing their Role plans. An organisation has a context that is described in terms of an ontology. Roles participate to the achievement of their organisation goals by means of their Capacities. In this subsection we will discuss the three most important elements of this domain: organisation, role, capacity. Definitions of the others can be found in Table 1 and on the ASPECS website.

An organisation is defined by a collection of roles that take part in systematic institutionalised patterns of interactions with other roles in a common context. This context consists in a shared knowledge, social rules/norms, social feelings, and it is defined according to an ontology. The aim of an organisation is to fulfil some requirements. An organisation can be seen as a tool to decompose a system and it is structured as an aggregate of several disjoint partitions. Each organisation aggregates several roles and it may itself be decomposed into sub-organisations.

[3] *Janus*: http://www.janus-project.org/

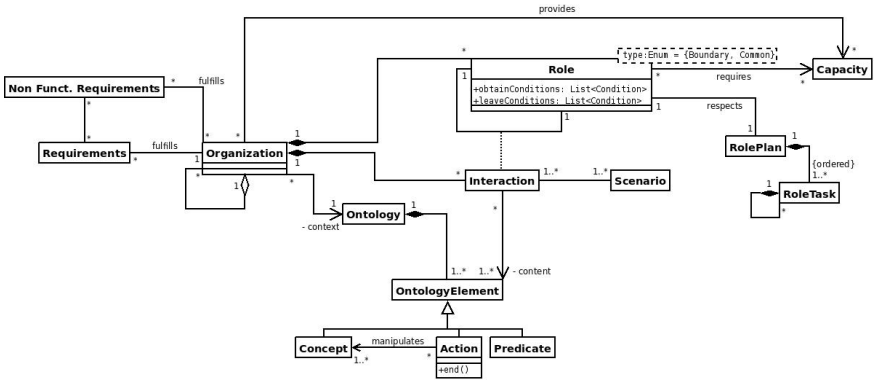

Fig. 2 The UML diagram of the ASPECS metamodel Problem Domain

In our approach, a Role defines an expected behaviour as a set of role tasks ordered by a plan, and a set of rights and obligations in the organisation context. The goal of each Role is to contribute to the fulfilment of (a part of) the requirements of the organisation within which it is defined.

In order to cope with the need of modelling system boundaries and system interactions with the external environment, we introduced two different types of roles: Common Role and Boundary Role. A Common Role is located inside the designed system and interacts with either Common or Boundary Roles. A Boundary Role is located at the boundary between the system and its environment and it is responsible for interactions happening at this border (i.e. GUI, Database wrappers, etc).

Roles use their capacities for participating to organisational goals fulfilment; a Capacity is a specification of a transformation of a part of the designed system or its environment. This transformation guarantees resulting properties if the system satisfies a set of constraints before the transformation. It may be considered as a specification of the pre- and post-conditions of a goal achievement. This concept is a high level abstraction that proved to be very useful for modelling a portion of the system capabilities without making any assumption about their implementations as it should be at the initial analysis stage.

A Capacity describes what a behaviour is able to do or what a behaviour may require to be defined. As a consequence, there are two main ways of using this concept: (i) it can specify the result of some role interactions, and consequently the results that an organisation as a whole may achieve with its behaviour. In this sense, it is possible to say that an organisation may exhibit a capacity. (ii) capacities may also be used to decompose complex role behaviours by abstracting and externalising a part of their tasks into capacities (for instance by delegating these tasks to other roles). In this case the capacity may be considered as a behavioural building block that increases modularity and reusability.

In order to complete the description of the possibilities offered by the application of our definitions of Organisation, Roles and Capacity, let us consider the need of modelling a complex system behaviour. We assume it is possible to de-

compose it from a functional point of view, and in this way we obtain a set of more finer grained (less complex) behaviours. Depending on the considered level of abstraction, an organisation can be seen either as a unitary behaviour or as a set of interacting behaviours. The concept of organisation is inherently a recursive one [19]. The same duality is also present in the concept of holon as it will be shown later in this article. Both are often illustrated by the same analogy: the composition of the human body. The human body, from a certain point of view, can be seen as a single entity with an identity, its own behaviour and personal emotions. Besides, it may also be regarded as a cluster/aggregate of organs, which are themselves made up of cells, and so on. At each level of this composition hierarchy, specific behaviours emerge. The body has an identity and a behaviour that is unique for each individual. Each organ has a specific mission: filtration for kidneys, extraction of oxygen for lungs or blood circulation for the heart. An organisation is either an aggregation of interacting behaviours, and a single behaviour composing an organisation at an upper level of abstraction; the resulting whole constitutes a hierarchy of behaviours that has specific goals to be met at each level. This recursive definition of organisation will form the basis of the analysis activities performed within ASPECS. In most systems, it is somewhat arbitrary as to where we leave off the partitioning and what subsystems we take as elementary (cf. [48, chap. 8]). This remains a pure design choice.

2.2.2 Agency Domain

The Agency Domain metamodel includes the elements that are used to define an agent-oriented solution for the problem analysed in the previous stage. By adopting an organisational approach, the solution will be mainly composed of the necessary social structures designed in a multi-perspective way. In this subsection we will discuss the most important elements of this domain. Definitions of the others can be found in Table 2 and further details on the ASPECS website.

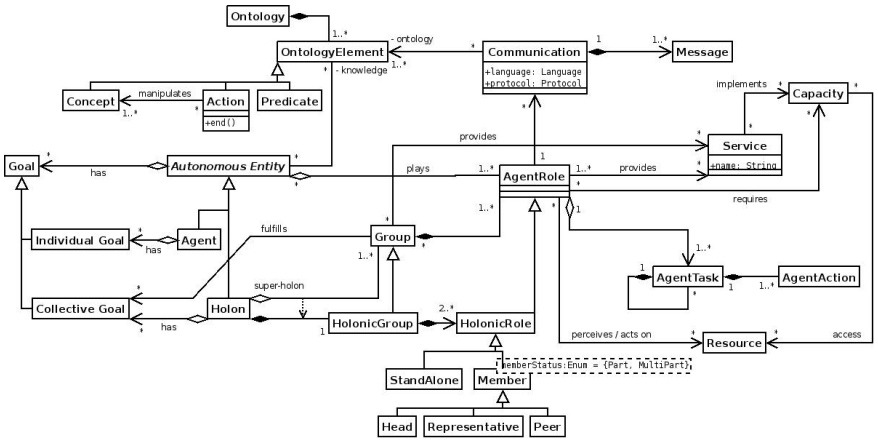

Fig. 3 The UML diagram of the ASPECS metamodel Agency Domain

Concept	Definition
Ontology	An explicit specification of a conceptualisation of a knowledge domain [28]. An ontology is composed of abstract ontology elements having three possible concrete types: **Concept**, **Predicate** or **Action**.
Concept	A category, an abstraction that shortens and summarises a variety/multiplicity of objects by generalising common identifiable properties.
Predicate	Assertions on concepts properties.
Action	A change realised by an entity that modifies one or more properties of one or more concepts.
Organisation	An organisation is defined by a collection of roles that take part in systematic institutionalised patterns of interactions with other roles in a common context. This context consists in shared knowledge and social rules/norms, social feelings, and is defined according to an ontology. The aim of an organisation is to fulfil some requirements.
Role	An expected behaviour (a set of role tasks ordered by a plan) and a set of rights and obligations in the organisation context. The goal of each Role is to contribute to the fulfilment of (a part of) the requirements of the organisation within which it is defined. A role can be instantiated either as a Common Role or Boundary Role. A Common Role is a role located inside the designed system and interacting with either Common or Boundary Roles. A Boundary Role is a role located at the boundary between the system and its outside and it is responsible for interactions happening at this border (i.e. GUI, Database, etc).
Interaction	A dynamic, not a priori known sequence of events (a specification of some occurrence that may potentially trigger effects on the system) exchanged among roles, or between roles and entities outside the agent system to be designed. Roles may react to the events according to theirs behaviours.
Capacity	A specification of a transformation of a part of the designed system or its environment. This transformation guarantees resulting properties if the system before the transformation satisfies a set of constraints. It may be considered as a specification of the pre- and post-conditions of a goal achievement.
Role Task	An activity that defines a part of a role behaviour. A *Role Task* may be atomic or composed by a coordinated sequence of subordinate *Role Tasks*. The definition of these *Role Tasks* can be based on capacities, required by roles.
Role plan	The behaviour of a *Role* is specified within a *Role plan*. It is the description of how to combine and order *Role Tasks* and interactions to fulfil a (part of a) requirement.
Scenario	Describes a sequence of role interactions, which fulfills a (part of) requirement.

Table 1 Definition of the problem domain concepts

Probably holon is the central element of the ASPECS design process. The term Holon was coined from the greek 'holos' meaning 'whole', and the suffix 'on' meaning 'part' or entity (for instance as a proton or neutron is a part of an atom); hence a holon is a whole to those parts beneath it in the hierarchy but at the same time a part to those wholes above it [37].

Each holon is an autonomous entity that has collective goals (shared by all members) and may be composed by other holons, called members or sub-holons. A composed holon is called super-holon. A super-holon is not only characterised by its members but also by their interaction patterns. This implies that two super-holons may be created from the same set of sub-holons if their members are interacting in a different way.

A super-holon contains at least one single *holonic group* to define how members get organised and how they govern the super-holon, and a set of *production groups* (at least one) to describe how members interact and how they coordinate their actions to fulfil the super-holon objectives. An example of a super-holon typical structure is reported in Figure 4.

Each super-holon member plays at least one role in the holonic group and various roles in production groups (at least one role in one production group). The *holonic group* describes the government of a holon and its structure in terms of authority/ power repartition. This group represents a *moderated group* (see [26]) 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 to the wide range of configurations it allows. Three *holonic roles* have been defined to describe the status of a member inside a super-holon and one role to describe the status of non-members:

Representative or holon interface: it is the 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). More than one member can play the *Representative* role at the same time.

Head or decision maker: it represents a privileged status conferring a certain level of authority in taking decisions inside the holon.

Peer or default member: Normally in charge of doing tasks assigned by *Heads*, a *Peer* 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.

Stand-Alone or non-member: This role represents a particular status inside a holonic system. In contrast to the previous holonic roles, it represents the way a member sees a non-member holon. *Stand-Alone* holons may interact with the *Representatives* to request their admission as new members of an existing super-holon.

The three first holonic roles describe the status of a member within a super-holon and participate in defining the holonic organisation. Each of these roles can be played by one or more members, knowing that any super-holon must have at least one *Representative* and one *Head*. The roles *Head* and *Peer* are exclusive between them, while *Representative* may be played simultaneously with one of the two others. Each of these member holonic roles is parameterised using a specific status that specifies if the corresponding holon member is shared between various super-holons. The *Part* status represents members belonging to only one super-

holon while the *Multi-Part* status represents sub-holons belonging to more than one super-holon.

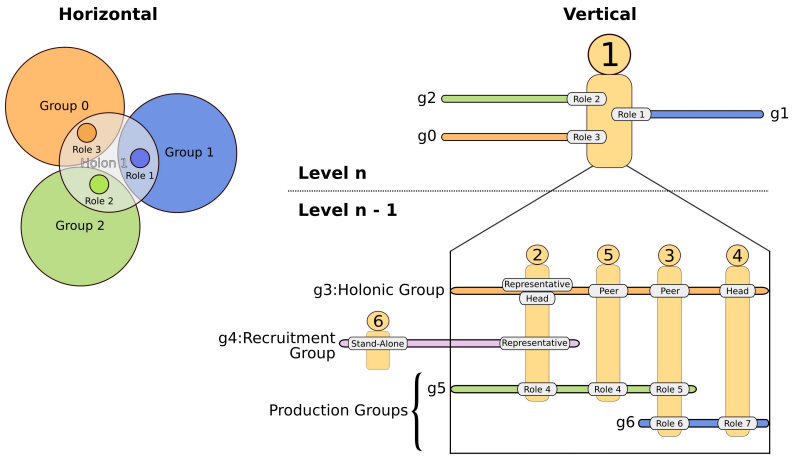

Fig. 4 Horizontal and vertical holon views

At the finest grained level of abstraction (that also means the first level of composition), holons are composed by groups and their associated roles are played by agents. From a motivational point of view agents and holons are different: an agent has *Individual Goals*, thus it is self-interested in reaching some goals. If the accomplishment of these goals prescribes or encourages the association to a holon, then the agent will try to join it and play one of the roles defined within it. On the other hand, a holon is motivated by *Collective Goals* that corresponds to a set of goals commonly shared among its members. A holon acts in the interest of the community of members that it embodies at an upper abstraction level.

The Agency Domain metamodel includes several other elements that because of space concerns we do not discuss here. Their definitions are reported in Table 2. In the next section we describe the elements of the Solution domain that are used to effectively code the solution designed with the elements described up to now.

2.2.3 Solution Domain The solution domain metamodel contains elements used for the implementation of the designed solution in the chosen platform. These elements are general enough to be applied to several existing platforms with minor or no changes but nonetheless the most suitable choice is *Janus* that directly inspired this portion of the metamodel.

JANUS (see [24]) was designed to facilitate the transition between the design and implementation phases of holonic systems. It is implemented in Java and it supports the direct implementation of the five key concepts used in the design phase: organisation, group, role, holon and capacity.

Organisation is implemented as a first-class entity (a singleton class in the object-oriented meaning of the word), which includes a set of role classes. An or-

Concept	Definition
Communication	An interaction between two or more roles where the content (language, ontology, and encoding) and the sequence of communication acts (protocol) are explicitly detailed. A *communication* is composed of messages expressing *communicative acts* [22, 23]. In a communication, participants are *Agent Roles* and the knowledge exchanged between them is explicitly represented by a set of ontology elements. A **Protocol** defines a sequence of expected message communicative acts and represents a common pattern of communication, a high-level strategy that governs the exchange of information between *Agent Roles*.
Group	An instance in the Agency Domain of an *Organisation* defined in the Problem Domain. It is used to model an aggregation of *Agent Roles* played by holons.
Agent Role	An instance of the Problem Domain *Role*. It is a behaviour (expressed by a set of Agent Tasks) and it owns a set of rights and obligations in a specific group context. *Agent Roles* interact with each other by using communications within the context of the group they belong to. Several *Agent Roles* are usually aggregated in the *Autonomous Entity* that plays them. An *Agent Role* may be responsible for providing one of more services to the remaining part of the society.
Holonic Group	A group that is devoted to contain *holonic roles* and takes care of the holon internal decision-making process (composed-holon's government). Holonic roles are used to represent in an organisational way the notion of moderated group (see [26]). They describe the level of authority of a member inside the holon members community and the degree of commitment of a member to its super-holon.
Agent Task	An *Agent Task* is a refinement of a Problem Domain *Role Task*. It is a portion of a role behaviour and it may be composed by other *Agent Tasks* or atomic *Agent Actions*. It may contribute to provide (a portion of) an *Agent Role*'s service.
Agent Action	The atomic composing unit of a behaviour. 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 *Agent Action* consists in invoking a capacity or a service requiring the same inputs.
Autonomous Entity	An abstract rational entity that adopts a decision in order to obtain the satisfaction of one or more of its own goals. An autonomous entity may play a set of *Agent Roles* within various groups. These roles interact with each other in the specific context provided by the entity itself. The entity context is given by the knowledge, the capacities owned by the entity itself. Roles share this context by the simple fact of being part of the same entity.
Agent	An autonomous entity that has specific individual goals and the intrinsic ability to realise some capacities.
Goal	A description of an objective to pursue and represents an abstraction of a projected state of affairs to obtain.
Individual Goal	A goal pursued by an individual agent that may be related to its personal desires or intentions. This agent will deliberate to determine a plan or a strategy to achieve its individual goals.
Collective Goal	A goal pursued by a community of individuals, which has the commitment of (a part of) the community members. Usually members commit to collective goals because achieving these goals contributes to the achievement of members' individual goals.
Service	It provides the result of the execution of a capacity thus accomplishing a set of functionalities on behalf of its owner: a role, a group, an agent or a holon. These functionalities can be effectively considered as the concrete implementation of various capacities. A role can thus publish some of its capacities and other members of the group can profit of them by means of a service exchange. Similarly a group, able to provide a collective capacity can share it with other groups by providing a service. A capacity is an internal aspect of an organisation or an agent, while the service is designed to be shared between various organisation or entities. To publish a capacity and thus allow other entities to benefit from it, a service is created.
Resource	The abstraction of an environmental entity. It may be manipulated by roles through specific capacities.

Table 2 Definition of the agency domain concepts

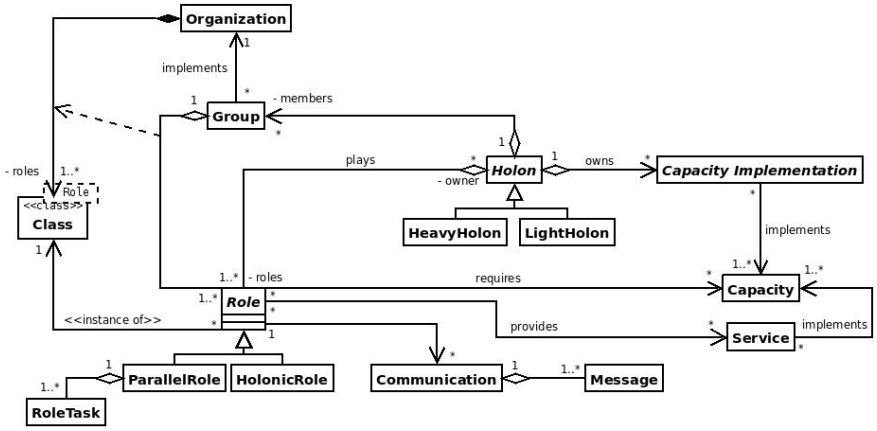

Fig. 5 The UML diagram of the ASPECS metamodel Solution Domain

ganisation can be instantiated in the form of groups. Each group contains a set of instances of different role classes associated with the organisation it belongs to. The number of authorised instances for each role is specified in the organisation description. One of the most interesting features of JANUS consists in the implementation of roles as first class entities. A role is seen as a full-fledged class, and each role is implemented independently of the entities that play it. Such an implementation facilitates the reuse of organisations in other solutions, but it also allows a wide dynamics for roles.

Janus defines two main types of holon: *HeavyHolon* (threaded) and *LightHolon* (non-threaded). A *HeavyHolon* has its own execution resources (one thread per holon), and it can therefore operate independently. The *LightHolon* is associated with synchronous execution mechanisms and it proved useful in some multiagent-based simulations. This approach resembles the synchronous engine of Madkit[4]. Holons can simultaneously play multiple roles in several groups; they can dynamically access to new roles and leave the ones that are no longer in use. When a holon starts playing a role, it obtains an instance of the class of this role that it stores in its role container; when it leaves a role, the corresponding instance is removed. To access or leave a role, a holon must meet the access and liberation conditions of the role and those of the corresponding group. This mechanism provides many advantages in terms of security, since holons have access to the behaviour of a role and thus get the corresponding executable code only if they fulfil these conditions. Agents are represented by atomic holons (non-composed ones).

The notion of capacity enables the representation of holons' competencies. Each holon has, since its creation, a set of basic skills, including the ability to play roles (and therefore communicate), to obtain information on existing organisations and groups within the platform, create other holons, and obtain new capacities. The capacity concept is an interface between the holon and the roles it plays. The role requires some capacities to define its behaviour, which can then be invoked

[4] see http://www.madkit.net/site/madkit/doc/devguide/synchronous.html

in one of the tasks that make up the behaviour of the role. The set of capacities required by a role are specified in the role access conditions. A capacity can be implemented in various ways, and each of these implementations is modelled by the notion of *Capacity Implementation*. This concept is the operational representation of the service concept defined in the Agency domain. Agents own the finest grained capacity implementations; these can be composed within the holons where these agents play roles in order to obtain more complex behaviours.

Because of space concerns, the complete definition of this metamodel elements are omitted but it can be found on the ASPECS website. The following section introduces a case study used to illustrate the various steps of the ASPECS development process and their associated notations.

2.3 Case study: Simulation of an Industrial Plant

A case study will be used throughout this paper for exemplifying the activities and artefacts composing the proposed design process. The case study deals with the analysis and modelling of one the most important industrial plant of eastern France. The plant belongs to a major automotive manufacturer, it is greater than 250 hectares, and, as most industrial plants, it is in perpetual evolution. The plant produces over 1700 cars per day and it requires constant improvements and expansions in order to handle the worldwide increasing demand of vehicles. As the production grows up new buildings need to be built and production units relocated. This plant, even including an internal railway, can be seen as a small town with a high traffic density. The plant counts on over 19000 employees working in different shifts to ensure the plant produces 24 hours a day. Last year an average of over 1600 trucks entered the plant every day carrying supplies. From a geographic point of view, three cities and a highway enclose the plant. Such a configuration makes difficult to increase the plant's size for accommodating new buildings, thus forcing to redesign the infrastructure when new needs arise.

Production chains are located inside buildings that exchange their products by using trucks. These trucks have predefined tours inside the plant. Buildings exchanging materials on a regular basis constitute a so-called Building Cluster. Identifying these clusters is very useful when planning trucks' routes and even more important when planning an infrastructure modification. Infrastructure modifications include (re)positioning the different workshops required by the material-processing plan inside existing buildings. Each workshop usually receives materials from outside the plant (raw materials) and/or another workshop. Materials produced by workshops go to other workshops or outside the plant (complete cars).

Any change in the location of a workshop could generate perturbations on the traffic flow and on the smooth functioning of the plant as a whole. Conversely, the proper positioning of workshops in existing buildings is an important optimisation activity that can result in an improved production for the overall plant.

Due to the great number of constraints and interrelated dependencies between traffic and production, a simulation tool could be of great help when evaluating different design choices. Even the smallest modification in a plant of this size

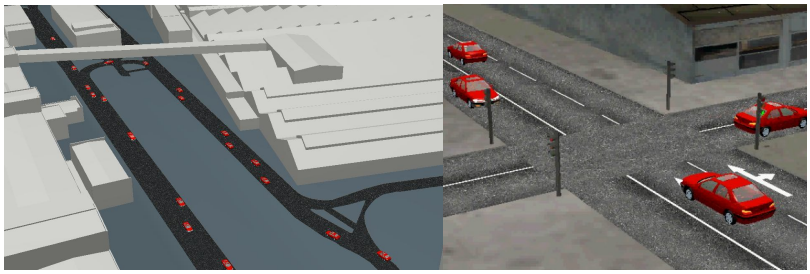

Fig. 6 Screenshots of the simulation

often requires a significant budget to be invested. A reliable simulator could offer the possibility of detecting side effects before the project approval.

Our simulation tool aims at providing a set of tools to support the decision maker in preparing infra-structural modifications (such as the construction of new buildings or parking lots, the relocation of workshops, etc) or when changing functional elements, like trucks' schedules and routes.

However, simulation of microscopic models may be inefficient when there are a great number of entities in the model. Moreover, multiple views of an unstructured model may be difficult to integrate. In order to deal with both these two problems we adopted the proposed holonic approach.

The simulator we developed offers a microscopic agent-based simulation of the industrial plant. It provides a set of indicators concerning congestion, jams, exchange of products between buildings. A connection with a Virtual Reality (VR) platform was realised to offer the possibility of visualising the simulation in real-time by using 3D technologies (see Figure 6).

This case study exhibits several properties that make it an ideal experiment for the evaluation of the ASPECS design process:

Openness: Trucks and cars enter and exit freely in the scenario. The number of involved trucks is not a priori known.

Complexity: Workshops are a priori positioned and their positions are fixed for each scenario. During the execution of each scenario, workshops are dynamically clustered in the Building Clusters, according to the volume of materials exchanged among them. Intuitively, the optimal solution consists in positioning tightly related workshops in the same building or in buildings that are nearby. The complexity of the problem arises from the fact that all workshops are some way related and therefore a modification in one single cluster may impact all the others.

Dynamics: The route of each truck is not a priori determined. The expected transit schedule has to be dynamically adapted to face delays, contingencies in production and traffic jams.

Workshop clusters are dynamically identified and may evolve according to the scenario events flow. This means that workshops may also enter and exit clusters at runtime.

In the following section we start detailing the ASPECS design process with the help of this case study. The description begins with a section reporting the activities of the *System Requirements* phase and then it continues with two sections describing the other two phases: *Agent Society*, and *Implementation and Deployment*.

3 System Requirements Analysis Phase

System requirements analysis phase aims at providing a full description of the problem based on the concepts defined in the *Problem Domain* of the metamodel. A complete description of this phase process can be found in the ASPECS website[5] and it is omitted here because of space concerns.

3.1 Domain Requirements Description (DRD)

Both the PASSI, and ASPECS, software processes are driven by requirements. Thus the starting activity deals with the analysis of system functional and non-functional requirements. Functional requirements describe the functions the software has to exhibit [1] 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 [1]. The global objective of the Domain Requirements Description (DRD) activity is gathering needs and expectations of application stake-holders and providing a complete description of the behaviour of the application to be developed. In the proposed approach, these requirements should be described by using the specific language of the application domain and a user perspective. This is usually done by adopting use case diagrams for the description of functional requirements; besides, conventional text annotations are applied to use cases documentation for describing non-functional requirements. In ASPECS, we advocate the use of a combination between use-case driven and goal-oriented requirements analysis where the description of functional requirements is completed by the one of associated goals and goal failures [11, 12]. The resulting document is labelled as the current activity: *Domain Requirements Description* (or briefly DRD) document.

Figure 7 details the use cases associated to a portion of the AMP case study. The single actor represents the manager of the manufacturing plant that will use this system as a decision support tool. The *manager* actor has two main requirements for the system: the first consists in identifying groups of buildings with important materials exchange (*Identify clusters* use case). The second requirement concentrates on the evaluation of vehicle traffic inside the plant (*Simulate traffic* use case). The goal is to identify and look at traffic related components, like roads, vehicles, traffic lights. This also requires modelling the topological structure of the plant. Traffic simulation also concerns the identification of parameters that will provide meaningful information to estimate congestion and possible jams.

[5] http://www.aspecs.org

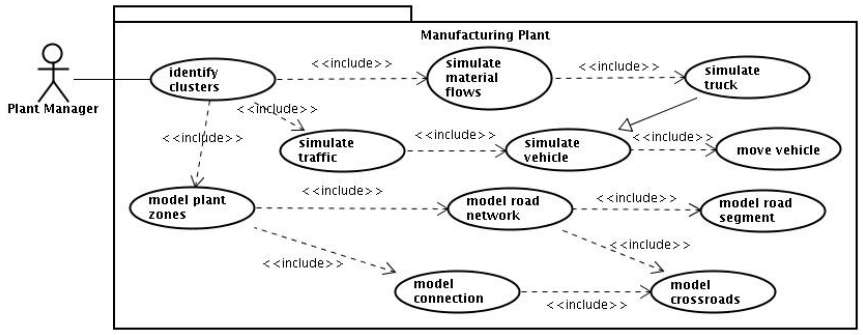

Fig. 7 Domain Requirements Description of the AMP decision support tool

3.2 Problem Ontology Description (POD)

The global objective of the Problem Ontology Description is to provide an overview of the problem domain. Stake-holders naturally express requirements in their own terms and with implicit knowledge of their own works [49]. 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 process and this has a direct consequence in the organisation and capacity identification activities. Problem ontology is modelled by using a class diagram where concepts, predicates and actions are identified by specific stereotypes. The POD consists in the conceptualisation of the requirements described in the previous activity and in any document that describes the system-to-be like, for instance, textual requirements. There exists several approaches for engineering ontologies such as linguistic studies, mining techniques or brainstorming. For a survey on ontology engineering see [**?**].

Problem Ontology of the AMP decision support tool is depicted in Figure 8, classes in grey are related to the solution ontology and will be discussed later in section 4.1. This ontology includes a *Plant* concept, which represents the entire plant. This is composed of *zones* that can be decomposed in smaller *zones* thus describing an area, which may be refined to the granularity of a *building* or a *road segment*. Each *zone* is linked to adjacent *zones* by *connections*. A *connection* can be refined in either a *gate* (if one of the two zones is a building or the road segment is located at the border of the plant) or a *crossroad*. *Vehicles* move on *road lanes* that compose the *road segments* and there is a specific type of *vehicle*, namely *Truck* that can carry materials between *buildings*; other *vehicles* such as cars and buses do not carry materials and they only contribute to traffic congestion.

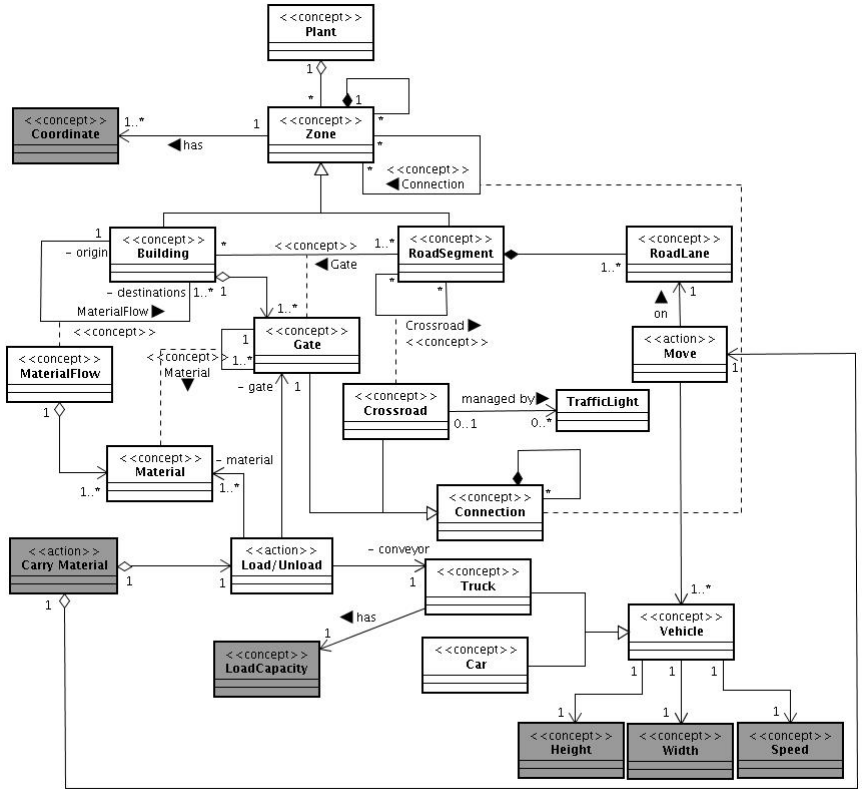

Fig. 8 Problem and Solution (in grey) Ontologies of the AMP decision support tool

3.3 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 a unique organisation (see Figure 2) in charge of fulfilling it. As already said, an organisation is defined by a set of 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.

Starting from use cases defined in the DRD activity, different approaches could be used to cluster them and identify organisations. We advocate the use of a combination between a structural (or ontological) approach mainly based on the analysis of the problem structure described in the POD and a functional approach based on requirement clustering.

Structural analysis focuses on the identification of the system structure. It is mainly based on the association between use cases and related ontological concept. In structural organisation identification, use cases that deal with the same ontological concepts are often put together in the same organisation. This approach assumes the same knowledge is probably shared or managed by the different mem-

bers of the organisation. The structure of the ontology itself can often constitute a good guideline to identify organisations, their composition relationships, and later their roles.

Behavioural analysis aims at identifying a global behaviour for the organisation intended to fulfil the requirements described in the corresponding use case diagram. The set of organisation roles and their interactions have to generate this higher-level behaviour. For this task, the use of *Organisational Design Patterns* [45] may be useful to the designer. In behavioural organisation identification, use cases dealing with related pieces of the system behaviour are grouped (for instance an use case and another related to it by an include relationship). This means that members of the same organisation share similar goals.

These two strategies are also used in methodologies such as GAIA [54]. On the one hand, the analyst can mimic the real world (through its conceptualisation in the ontology) if its structure is a mandatory or relevant aspect of the system-to-be. On the other hand if the existing organisations are not efficient nor relevant for the system-to-be the use of a behavioural point of view is probably to be preferred. In the two cases, the subsequent activities determine if the choices made are consistent. Indeed, the identification of roles and interactions (cf. section 3.4) and their refinement through scenarios descriptions (cf. section 3.5) and role plans (cf. section 3.6) validate the assignments (fulfil relationships) of organisations to use cases.

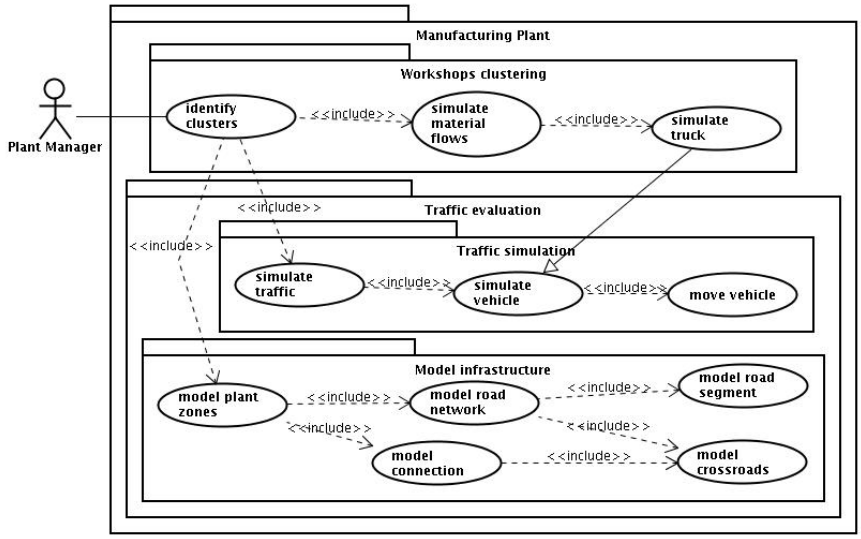

Fig. 9 Fragment of the Organisation Identification of the AMP decision-helping tool

The use case diagram presented in Figure 9 presents a part of the organisation identification diagram. For instance, use cases *Simulate Traffic*, *Simulate Vehicle*

and *Move Vehicle* are clustered in the *Traffic simulation* organisation according to a functional identification of the resulting *Traffic Simulation* organisation.

3.4 Interactions and Role Identification (IRI)

The Interactions and Role Identification (IRI) activity aims at decomposing a global behaviour embodied by an organisation into smaller interacting behaviours. Each of these finer grained behaviours will be exhibited by a Role. Interacting roles must be defined in the same organisation that provides the interaction context. The goal of each Role 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 perimeter definition started in the domain requirements description activity. This is done by adopting two different types of Roles: Common Role (often called just Role) or Boundary Role. This latter has been conceived to work at the borders of the designed system. Boundary Roles can be, for instance, used to control external sensors/effectors or to interact with other systems. The result is a class diagram where classes represent roles (stereotypes are used to differentiate common and boundary roles), packages represent organisations and relationships describe interactions among roles or contributions (to the achievement of a goal) from one organisation to another. Performing this activity is usually an iterative process coordinated with the following Scenarios Description. In this latter, elements that have here been depicted from a structural point of view are exploited in their dynamical behaviour.

Some methodological guidelines may be provided to explicit this iterative process: the first step consists in looking into scenarios that can be deducted from use case diagrams to identify interactions. Let us suppose, for instance, that organisation O_1 is assigned to fulfil requirements represented by use cases A and B. Use case B has an "*include*" relationship with use case C assigned to organisation O_2. This encourages the designer to explore the possibility of a scenario where a role of O_2 interacts with another role of O_1 in order to provide to O_1 the result of a capacity that belongs to O_2. Another guideline for roles identification consists in looking at the structure of the ontology in order to find elements that suggest some hierarchical structure that could evoke a holonic configuration with interacting roles (usually such a configuration is also useful for organisations identification as it has been said before). The process stops when all organisations have been decomposed into interacting roles and when all corresponding interactions have been described in at least one scenario. Organisations fulfilling requirements with mutual dependencies in the use case diagram should be linked during the IRI activity, usually by a "*contributes to*" dependency.

We will now focus on the *Traffic simulation* organisation of the AMP case study that is located at this finest level of granularity in the hierarchical decomposition of the plant. This organisation can be decomposed in three roles (see Figure 10): *Road User*, *Crossroad* and *RoadSegment*, which are a Common Role. However *Crossroad* and *RoadSegment* are located at the boundary between the plant and the outside region and they can be regarded as Boundary Roles. A *Road User*

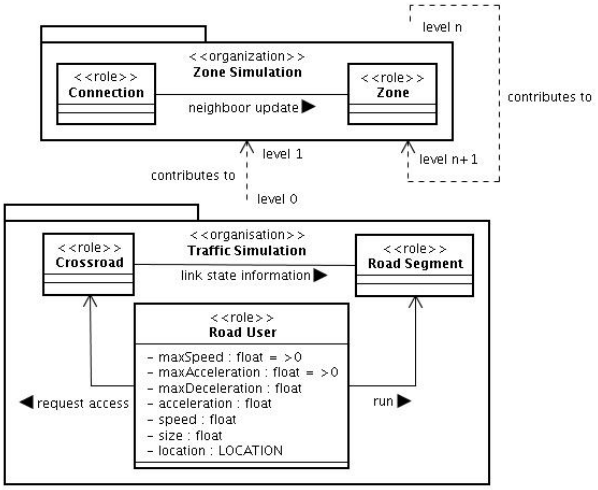

Fig. 10 Fragment of the Interactions and Role Identification for the AMP decision-helping tool

can drive along a *RoadSegment* and cross a *Crossroad* if environmental conditions allow it (for instance no other *Road User* is crossing a *Crossroad* at the same time).

The *Zone simulation* organisation corresponds to the other levels of granularities in the hierarchical decomposition of the plant. This organisation can be recursively used to decompose behaviours until we reach the lowest one that is modelled by the *Traffic simulation* organisation. The *Zone simulation* organisation is composed of two roles: *Connection* and *Zone*. As stated in the ontology, a *Zone* can be decomposed in smaller *Zones* (and finally decomposed in *Road Segments* and *Crossroad* at the lowest level). The simulation of a *Zone* can then be the result of the simulation of smaller *Zones*. This contribution relationship is also depicted in Figure 10 by using a UML constraint named "*contributes to*".

3.5 Scenario Description (SD)

The goal of this activity is to describe the sequence of interactions occurring among roles involved in each scenario. Scenario description is done just after *OID* and *IRI* activities, and at this stage it is possible to assign an organisation and a set of interacting behaviours (enacted by involved roles) to each requirement. The challenge now consists in the description of how these different roles are interacting to realise the scenario. In this sense, the required scenarios can be seen as a conventional design activity dealing with capturing the behaviour of the system in the most relevant occurrences of use cases (scenarios are also often referred to as instances of use cases). Designed scenarios should describe real examples of program execution and they should also include a description of the normal event flow [49]. Scenarios are drawn in form of UML sequence diagrams and participat-

ing roles are depicted as object-roles. The role name is specified together with the organisation it belongs to.

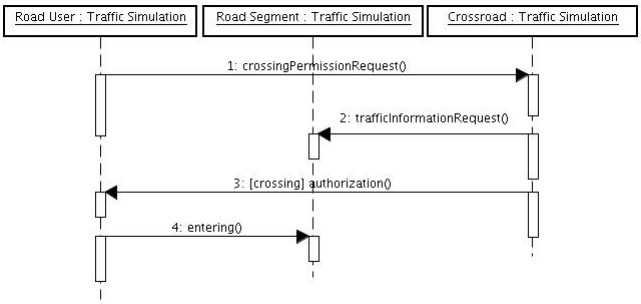

Fig. 11 A scenario description for the AMP Traffic Simulation Organisation

In order to perform this activity, a suggested guideline consists in starting from the system behaviour as it has been described in the Organisation Identification activity (see subsection 3.3). There, the system behaviour had been specified in terms of use cases, it had been partitioned by using packages, and finally assigned to responsible organisations. Besides, it is worth to note that Scenario Description activity is tightly related to the preceding one (Interactions and Role Identification) where the roles populating scenarios are depicted from a statical point of view together with their relationships.

For the already proposed *Traffic simulation* organisation the scenario described in Figure 11 corresponds to a *Road User* trying to cross a *Crossroad*. First it requests permission to the *Crossroad*. The *Crossroad* first checks if the required crossing is compliant with local traffic rules (prescribed directions, obligatory turns, etc) and then requests traffic information to the next *Road Segment* (the destination one). If entering the new *Road Segment* is possible then the *Crossroad* grants the permission to the *Road User*, which crosses it.

3.6 Role Plan (RP)

The goal of each Role is to contribute to fulfil (a part of) the requirements of the organisation within which it is defined. The behaviour of a Role 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 interactions, external events, and Role Tasks in order to fulfil a (part of a) requirement. A Role Task is the specification of a parameterised behaviour in form of a coordinated sequence of subordinate units (a Role Task can be composed of other Role Tasks).

The first task in this activity consists in detailing responsibilities assigned to the currently designed role. For each role, a set of Role Tasks has to be identified

for accomplishing the assigned requirements. Roles interactions that have been already defined in previous activities may prove useful in the definition of the plan and at the same time they are constraints to be satisfied by the plan (the role has to engage in the already identified interactions in order to exhibit a behaviour that is coherent with the scenarios it is involved in). 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 in the next activity and the role will refer to it.

The resulting work product is an UML activity diagram reporting one swimlane for each role. Activities of each role are positioned in its swimlane and interactions with other roles are depicted in form of signal events or object flows corresponding to exchanged messages.

Figure 12 reports an example of a Role Plan diagram. It depicts the plan of the *Traffic simulation* organisation starting with an external signal, which indicates the beginning of the simulation. After that, the *Road User* establishes a route and the corresponding motion variables according to the chosen route; it follows on until a *Crossroad* is reached. When the *Road User* reaches a *Crossroad*, it asks the *Crossroad* for a crossing permission. The *Crossroad* grants the permission if the *Crossroad* is available and if the *Road Segment* destination is not jammed. The *Crossroad* sends a suggestion to the *Road User*, which takes the final decision. When the *Road User* exits the *plant* it is destroyed, if it does not and if the *Crossroad* is free then the *Road User* crosses it and informs the new road of its entrance. When the *Crossroad* is busy, the *Road User* waits. The plan of the *Road Segment* consists in waiting for information requests (for instance about traffic) and incoming *Road Users*.

3.7 Capacity Identification (CI)

The main objective of the Capacity Identification activity (CI) is the definition of generic role behaviour by identifying which know-how a role requires from the individual that will play it. As already said, a 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. It means that the results described by a capacity may be reached by adopting different strategies. The realisation of the capacity is a concern of the Agency Domain and it will be discussed later.

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. The final result of this activity (see Figure 13) is performing a revision of the already designed IRI diagram by adding capacities (represented by classes) and relating them to the roles that require them.

In Figure 13, roles of the *Traffic Simulation* and *Zone Simulation* organisations are linked to three capacities. An entity playing the *Road User* role must own the *Choose Route* capacity. This capacity allows the computation of a route between two points according to environmental constraints. Entities playing the *Connection*

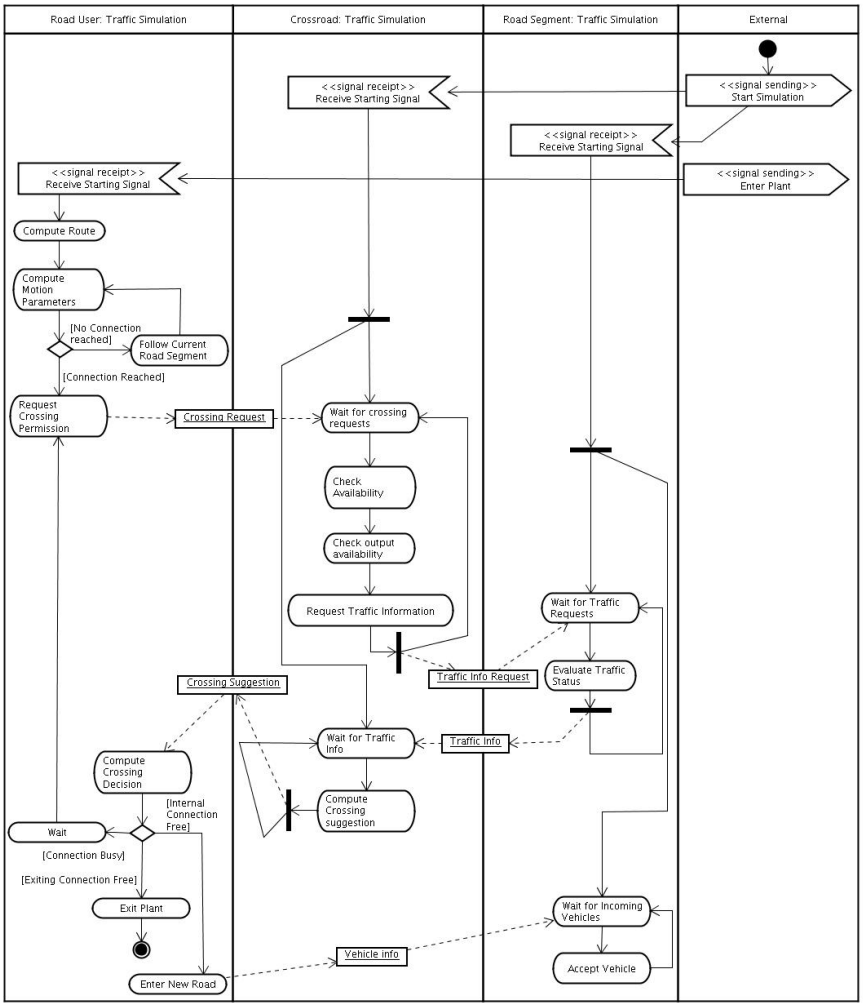

Fig. 12 Roles Plan of the AMP Traffic Simulation Organisation

and *Zone* roles must have a zooming capacity (for enabling the decomposition in smaller zones).

At the end of the Capacity Identification activity, a set of capacity have been identified. If the complexity of these capacities is manageable by atomic easy-to-implement entities, the iterative analysis process stops. Otherwise, the complex capacities are considered as new requirements, and a new iteration starts. The System Requirements Analysis phase needs to be reiterated to consider these new requirements and thus identify organisations in charge of realising them, and so on. More details on this aspect may be found in [**?**].

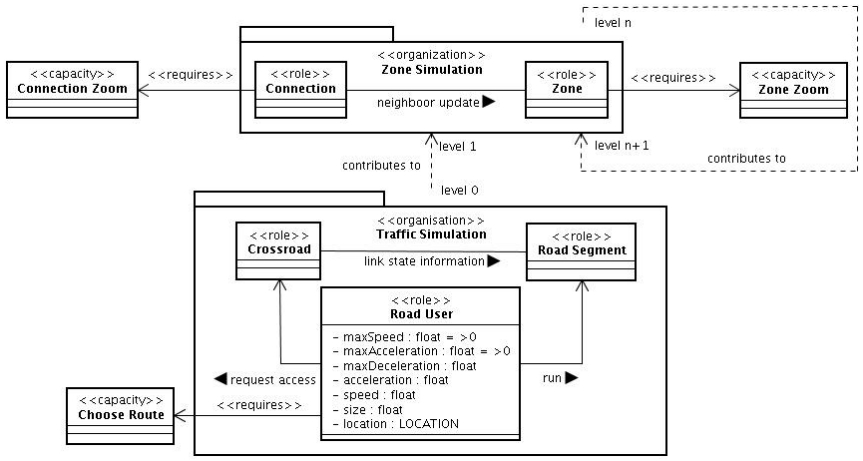

Fig. 13 Capacity Identification of the AMP Traffic and Zone Simulation Organisations

4 Agent Society Design Phase

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 has been modelled in terms of organisations, roles, capacities and interactions. The result of this design phase is a model of the agent society involved in the solution in terms of social interactions and dependencies among entities (Holons and/or Agents).

After the first activity (Solution Ontology Description), the design flow is split in two alternate paths: one concerning the social and organisational aspects of the system and the other dealing with the design of agents considered as individuals with their own individual goals. The resulting agents will be positioned at the lowest level of the hierarchical social structure while holons will cluster roles played by them thus building the holarchy.

In other words, agents are designed as individuals owning the capacity implementations necessary to play roles composing social structures (holons). In turn, once obtained (by agents) the realisation of the capacities necessary to exhibit their behaviours, holons can be considered at a higher level of abstraction as capacity implementation owners by themselves, and therefore they can play roles in higher level holons thus enabling the composition of the holarchy.

In the following sub-sections each activity will be detailed as it has been done for the System Requirements Analysis phase activities.

4.1 Solution Ontology Description (SOD)

The objective of this activity consists in refining the problem ontology described during POD activity by adding new concepts related to the agent-based solution

and by refining the existing ones. Concepts, predicates and actions of this ontology are now also intended to be used for describing information exchanged in communications among roles. This implies the definition of all the predicates that are used to exchange knowledge in communications as well as the actions that can be done by Holons/Agents and affect the status of the world they live in (as represented in ontology by concepts). The introduction of actions in ontology is not new and is also compliant with a FIPA[6] specification (RDF [21]).

The presence of actions in the ontology allows to model the complete knowledge space of autonomous entities, in terms of the concepts they can understand, the predicates they can assert about the status of those concepts, and the actions they can perform/conceive in order to affect the status of concepts.

This activity follows an iterative and incremental design approach. The need for new concepts, predicates and actions can arise at any moment in the design activities and can justify iterations to improve ontology with the new elements.

As regards the proposed case study, in order to fully support the physical model of vehicles moving in the plant, the ontology (see figure 8) as been enriched with concepts concerning features of a *Vehicle* such as *Height*, *Width*, *Speed* or *Load-Capacity* for a *Truck*. Each *Zone* is now defined by a set of coordinates.

At the end of the Solution Ontology Description activity, the ASPECS development process is split up into two development sub-branches, the first and foremost is dedicated to the organisational design of the system, the second is dedicated to the identification and design of agents composing the system. In other words, the main branch deals with the design of the organisational structure of the system and collective goals that have to be satisfied by organisations, while the second branch deals with agent's personal goals and motivations, and it aims at defining the agent architecture. The two branches are then merged to describe the complete holarchy structure of the system and the individual agent decision plan. In the following subsections, the activities related to agents design and then the organisational ones will be described. This corresponds to building the holarchy in a bottom-up way. This is not a prescription of the proposed approach but only a presentation choice. The designer is free of choosing his/her preferred branch and even (most likely) interleaving the activities of the two paths. The Agents Identification activity is the first activity of the agent design and it will be discussed in the next section.

4.2 Agent Identification (AI)

This Agent Identification (AI) activity consists in identifying agents that will compose the lowest level of the system hierarchy and their responsibilities. These responsibilities are modelled using the notion of individual goals and will be the basis to determine agent architectures in the next activity. The *Interactions and Role Identification*, *Solution Ontology* and *Domain Requirements Description* Documents are the main inputs of this activity. Agent's goals identification is mainly based on gathering organisation responsibilities located at the lowest level of the system organisational hierarchy. These responsibilities are expressed in term of

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

requirements described by using a combination between a use-case driven and a goal-oriented approach [11, 12]. Agents are conceived to play these lowest-level roles; their personal goals should thus at least correspond to the union of the goals of these roles. To play these roles, agents have also to provide an implementation for the capacities required by these roles. This aspect will be studied in the next activity. Besides, Agent Identification activity is also guided by the identification of ontology concepts that represent system individuals (concepts linked to ontology actions, for instance *Truck* in our case study). These latter are effectively considered as useful guidelines to identify agent responsibilities since an individual acts according to personal motivations and goals.

We propose to use TROPOS goal and actor diagram to describe the results of this activity. However, system and agent overview diagrams as proposed in PROMETHEUS [43] may also be used as an alternative solution.

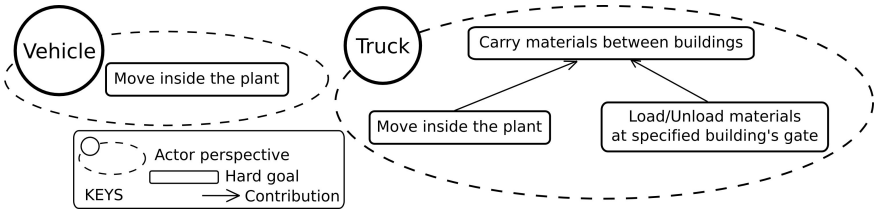

Fig. 14 Agent Identification for the AMP Case study

Figure 14 describes the *Vehicle* and *Truck* agents of the AMP Case study and their respective responsibilities using a TROPOS Goal diagram.

4.3 Agent Architecture Description (AAD)

The Agent Architecture Description (AAD) activity aims at providing precise indications on the architecture that should be adopted by agents. Indeed, the agent architecture is at least defined by the set of roles that the agent should play and the minimal set of services that implement the capacities required by these roles. The association between Agents and Agent Roles allows the identification of the set of capacities that are required by Agent Role in order to be played by Agents. In this activity, a UML class diagram is used to describe agents and their capacities realisations in terms of attributes and methods.

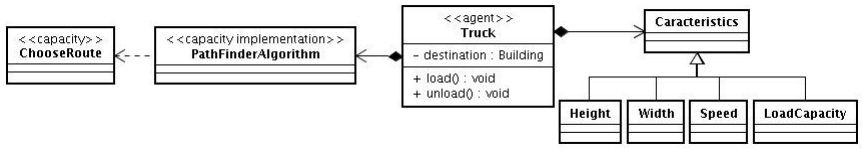

Fig. 15 Description of the Truck Agent architecture

The *Truck* agent architecture sketched in figure 15 consists mainly in an implementation of a *Path Finder* Algorithm, which realises the *Choose Route* capacity.

4.4 Communication Ontological Description (COD)

This activity aims at describing communications among roles. A communication is an i nteraction between two or more roles where the content (language, ontology, and encoding) and the sequence of communication acts (protocol) are explicitly detailed. A communication mainly consists of speech acts and protocols as also specified by FIPA. The model of communication adopted 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 communication is an interaction composed of several messages ordered by a Protocol. Each message underpins a specific communicative act (see [46] and FIPA speech acts [22, 23]) and its content refers to one or more ontology elements. The message is encoded in a content language.

At this stage we could regard the previously studied interactions as messages and each set of interactions between two roles has to be clustered in one or more communications. This activity also describes data (it would be better to say *knowledge*) structures required in each role to store exchanged information by adding the necessary ontological structure to roles. These structures are of course based on the elements of the solution ontology.

Figure 16 describes some communications of the AMP Traffic Simulation Organisation. For example, each *Road User* role-player may initiate an *Entering* communication ruled by the *FIPA-inform* protocol, using the Solution Ontology previously described and encoded in RDF.

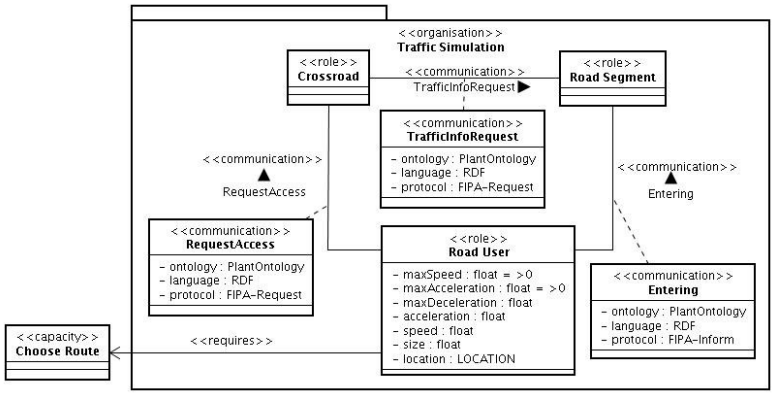

Fig. 16 Communication Ontological Description of the AMP Traffic Simulation Organisation

4.5 Role Behaviour Description (RBD)

This activity aims at defining the complete life-cycle of a role; Roles identified during the IRI activity are here specialised in Agent Roles, which interact with each other by means of communications. The behaviour of Agent Roles is described by a set of Agent Tasks that are the refinement of the Problem Domain Role Tasks and contribute to provide (a portion of) an Agent Role's service. At this level of abstraction, this kind of task is no more considered atomic but it can be decomposed in finer grained Agent Actions.

An Agent Action is now the atomic unit of a 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 Agent Action consists in invoking a capacity or requesting a service (as explained in following subsections).

The Role Behaviour Description is a refinement of the results produced by the Role Plan activity performed in the 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 their expressiveness, their executability and their capabilities to generate code. If a role requires 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.

Figure 17 describes the behaviour of the *Road User* Role. By default it is *idle* just the time for computing a route (the transition without event). Once the route is computed it starts travelling on a *road lane*. This state remains active until a *crossroad* is reached. When such an event occurs the role sends a *requestCrossingPermission* message and enters the waiting state. When the *crossingPermission* reply is received the role either continue in the *running on a road lane* state if it is still in the plant or exits the plant. If no answer is provided before a timeout, the *Road User* computes a new route.

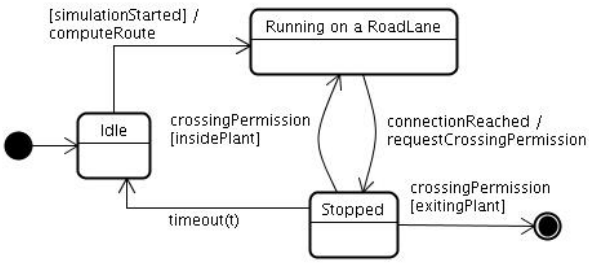

Fig. 17 Role Behaviour Description of the *Road User* role in the *Traffic Zone* Organisation

4.6 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 does not match any of the existing FIPA protocols.

The designer starts from the scenarios and the ontological description of communications in order to find if an existing protocol can be used. If not, then he/she can proceed to the definition of a new protocol that is compliant with the interactions described in scenarios and communication semantics.

It is advisable to refer to the FIPA Interaction protocols library[7] 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.

For our case study there is no need of designing new protocols as we reused existing ones.

4.7 Organisation Dependencies Description (ODD)

The goal of the Organisation Dependencies Description activity is to define relationships between: (i) capacities required by roles and organisations, and (ii) services that realise them. It also dedicated to the identification of resources.

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 from 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 on the real world are made explicit.

Finally, this activity should also outcome with the description of interfaces used by the system to manipulate resources. This matching between service and capacity allows the construction of a repository that may be used to inform agents on how to dynamically obtain a given capacity. Moreover it also proves that the hierarchical system decomposition is correct since the matching should validate the contribution that organisations acting at a given level give to upper-level organisations.

The resulting work product, as exemplified in Figure 18, is a UML class diagram, reporting roles (clustered in organisations), communications, services, capacities and resources. It can be seen as a refinement of the COD (Communication Ontological Description) diagram including services and resources. Figure 18 describes dependencies of the *Traffic Simulation* organisation. One new resource has been identified (it represents a 3D virtual engine), and the *RenderVehicle* 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

[7] FIPA Interaction Protocols specifications: http://www.fipa.org/repository/ips.php3

the *Road User* role that does not need to publish it as a service. It is the same for the *Choose Route* capacity. The *Connection Zoom* and *Zone Zoom* capacities have service realisation since the zooming is done by mean of sub-holons contribution.

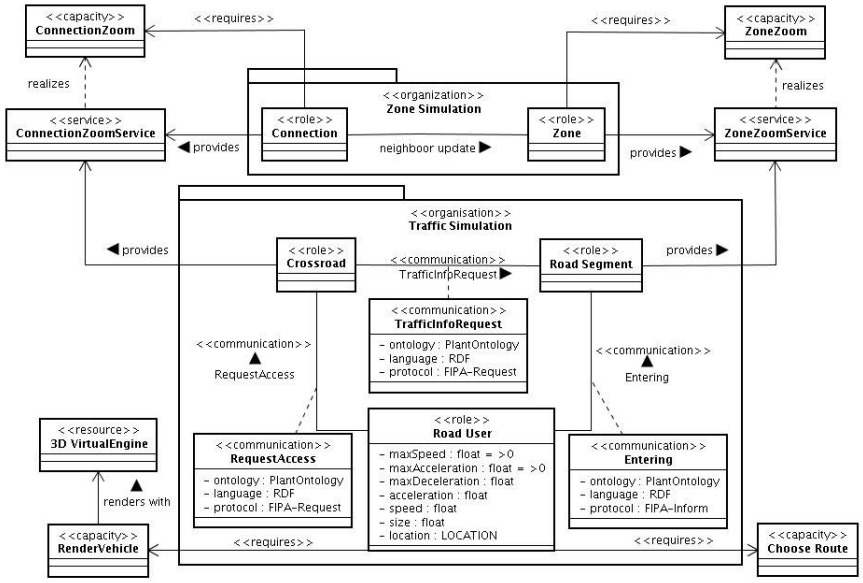

Fig. 18 Organisation Dependencies Description of the AMP Traffic and Zone Simulation Organisations

4.8 Role Constraints Identification (RCI)

This activity aims at identifying constraints between roles. This for instance includes roles that have to be played simultaneously, priorities in their executions, mutual exclusions, dependencies, 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 mean to realise 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 by other organisations and it thus allows the identification of roles couples that have to be played simultaneously.

In the presented case study, if an agent plays the role *Carrier*, it must play at the same time the role *Road User*. This type of constraints is modelled by using a stereotyped UML dependency from the *Carrier* and *Road User* classes. The direction of the dependency means that the *Carrier* role required that its player already plays the *Road User* role.

4.9 Agent Plan Description (APD)

This activity aims at terminating the design of agent internal architectures. According to the results of the Agent Architecture Description and Role Constraints Identification activities, it is now possible to determine the personal plan of each agent according to its individual motivations and pursued goals. In this activity, each agent of the system is associated to the set of roles it has to play according to the set of capacities it owns. An agent has to provide an implementation for each capacity required by the played roles. The plan represents the strategy used by the agent to choose the roles it plays. In this activity, a state-chart diagram is used to describe the plan of an agent; an activity diagram may also be used.

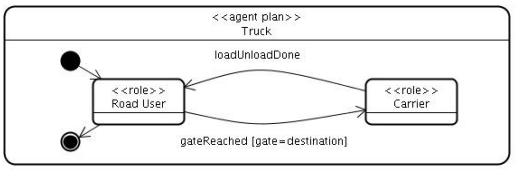

Fig. 19 Description of the Truck Agent Plan

The *Truck* agent plays two roles, namely *Road User* and *Carrier*. By default it plays the *Road User* role and if it reaches a Gate it decides to play the *Carrier* role. When the load/unload operations are finished it returns playing the *Road User* role.

4.10 Holarchy Design (HD)

At this step in the development process, the set of organisations composing the system, their roles and the associated communications have been identified and specified. The architectures of the various agents have also been specified. The Holarchy Design activity is the last activity of the Agent Society design phase and aims at providing a global synthesis where previous activities work products are combined and summarised in a single work-product describing the overall structure of the system and the rules that will govern its dynamics.

In order to properly define the discussed aspects of each holon, the Holarchy Design activity is decomposed in four main tasks that are detailed in what follows.

Holonification task. This task aims at mapping the previously identified hierarchy of organisations to a holarchy. This mapping is based on the association of holons composing the holarchy with the set of roles defined in the organisation hierarchy they have to play. To build holarchies, organisations that composed the system are instantiated in form of groups. A set of previously identified agents composes the lowest level of the holarchies. A set of holons is then created at each upper level, each holon may play one or more roles in one or several groups in the level of interest. Composition relationships among super- and sub-holons are then specified according to the contributions required by the organisations (as described in the OID and ODD work products).

In this activity, two points of view on the system are used to conceive the final system holarchy. Each of these viewpoints corresponds to a dimension of the holon concept (see Figure 4):

Horizontal: This step consists in instantiating organisations of the same level in terms of groups. Then, holons will be created for clustering these groups and they will be associated to the roles they should play according to the results of the ODD and RCI activities.

Vertical: This step aims at specifying the composition relationship between holons. It specifies how a group of holons of level n will contribute to the behaviour of a role played by a holon of level $n + 1$. Groups of level n are instances of organisations that provide services able to implement capacities required by roles located at level $n + 1$.

Holon Government Model Definition task. The second task focuses on newly composed holons and it 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. Defining the holon government type essentially means defining the holon 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 process that has to be defined (for instance, a voting mechanism may be used).

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) who is to be contacted by the external holon that wants to enter the super-holon or that is requesting a service and how the requesting process could be started.

The decision process for the admission of a new member is an example of decision process that fits most of the cases and for this reason we will mainly refer to that without loosing in generality. The decision can be done according to several different internal policies representing different levels of involvement of the holon member 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 loose most of their autonomy having to request the head permission for providing a service or requesting a collective action. Another possibility consists in establishing a voting mechanism.

Specific and interesting configurations can arise from the number of voters and the percentage of *heads* and *peers* involved in the decision-making process, because of their relevance it is worth to analyse them in details:

Monarchy: the command is centralised and a Head is in charge. Monarchy, here, doesn't refer to the process of Head's nomination/election. The nomination process is a different issue from the decision-making process. Monarchy here describes the situation where only one head controls the entire decision-making process.

Oligarchy: A little group of heads share the command without referring to the other (peer) members.

Polyarchy [8]: A little group of heads share the command but they have to refer to the *Peers* for some decisions.

Apanarchy [9]: The command is completely shared between all members of the super-holon. Everyone takes part to the decision-making process.

Holarchy Definition task. 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. Results of this task are summarised in an organisational cheese-board (see Figure 20) that is an extension of the cheese-board diagrams introduced in [20]. This diagram is then associated to a set of documents describing the government of each holon and the rules governing their dynamics as above discussed.

Holon self-Organisation Mechanisms Definition task. 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.

Because of space concerns, only the most common and important rules governing holon dynamics are discussed here, mainly those dealing with members' recruitment and holon creation.

Once a super-holon has been created, new members may request to join it or the super-holon itself may recruit new members to achieve its own goals. 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 for submitting their admission request. A specific organisation with two roles, *StandAlone* played by the candidate, and *Representative* played by at least one of the representatives of holons members, has been designed to manage this recruitment process.

As regards the holon creation mechanism, it is important to study the motivations for the birth of a new holon; these can in fact either depend: (i) on the need

[8] We borrow the term coined by Robert A. Dahl [16] to describe a specific type of democratic government.

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

to satisfy in a collective way a requirement that cannot be accomplished by a single entity alone, or (ii) on the need to improve the internal structure of an existing holon that is becoming too big and whose tasks are too complex to be managed. It is therefore possible to distinguish two different mechanisms:

A top-down mechanism (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 the organisation complexity. This case could be reduced to a specific one of the initial creation process, because newly created holons are configured to satisfy integration constraints with the super-holon.

A bottom-up mechanism (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 new super-holon have to be defined.

A fragment of the final structure of the holonic solution for the AMP case study is presented in Figure 20 using a holonic cheese-board diagram. This diagram is associated to a map describing two levels of the associated topological decomposition of two plant zones that are modelled in the application by holons 1 and 2. At the second level of the holarchy, three super-holons (1, 2 and 3) are playing roles in two groups $g0$ and $g1$. The denomination $g0$*: Zone Simulation* indicates that group $g0$ is an instance of the *Zone Simulation* organisation. Holons 1 and 2 represent two plant zones that are linked using a connection embodied by the holon 3 who maintains statistic information about material flows between the two adjacent zones. Each of these super-holons contains at least one instance of the *Traffic Simulation* organisation ($g3$, $g5$ and $g6$) in charge of the simulation of trucks and vehicles traffic inside the zone and it also contains a holonic group defining the governmental structure. Each *Zone* holon disposes of a simple type of government inspired by the oligarchy model where command is centralised in the hands of a group of heads. The rule is that the holon playing the *Road Segment* role is automatically promoted Head and all heads elect one Representative among them. The agent 7 is shared by two super-holons (1, 2) and thus considered as a Multi-Part Peer. This holon constitutes the way to transfer vehicles between the two zones represented by holons 1 and 2.

5 Implementation and Deployment Phase

This section gives an overview of the Implementation and Deployment phase. As already said, further details can be found in [24]. This phase aims at implementing and deploying the agent-oriented solution designed in the previous phase by adapting it to the chosen implementation platform.

A platform called *Janus*[10] 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.

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

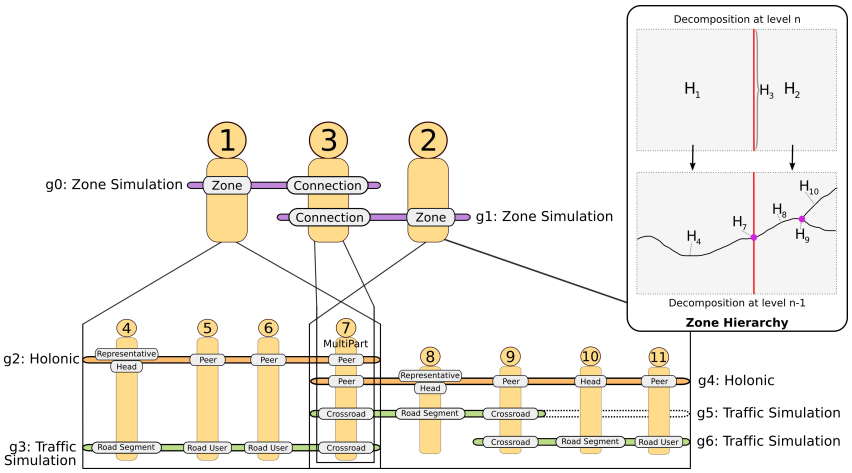

Fig. 20 A fragment of the Holarchy Design of the AMP decision-helping tools

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 [30], JADE, and FIPA-OS, the concept of Role is considered as a first class entity in *Janus*. It thus enables a direct implementation of organisational models without making any assumptions on the architecture of the holons that will play the role(s) of an organisation.

Based on *Janus*, the implementation and deployment phase activities allow the description of the solution architecture and the production of associated source code and test. This phase also aims at detailing how to deploy an application over various *Janus* kernels. *Janus* adopts a peer-to-peer technology to allow kernel federation and agent migration. Of course, the process described in this phase can also be used with any other platform able to provide a translation of the concepts presented in the ASPECS metamodel of the Solution domain.

5.1 Holon Architecture Definition

This activity aims at defining the architecture of each holon involved in the implementation of the previously designed solution. Each organisation together with its set of roles and associated tasks has to be described. Each holon is associated with the set of roles it should play, the set of capacities and services it owns. Two different approaches may be used to design a holon. A static approach consists in designing a specific architecture for each holon during the Holarchy Design activity. This approach is the simplest and easiest to maintain, but it may generate a relevant number of different architectures in complex applications. Another approach consists in designing a dynamic 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.

Figure 21 depicts a part of the Holons architecture defined for the implementation of our AMP example. Environmental parts were implemented as *LightHolons* (non-threaded). The *Car* and *Truck* agents are implemented as *HeavyHolons* (threaded). The *Traffic Simulation* and *Workshop Clustering* organisations are partially described in Figure 21. Only the role *Car User* is described in terms of its *RoleTasks*. Classes in grey correspond to classes of the solution domain of the ASPECS metamodel (see Figure 5) that are refined to introduce problem dependent artefacts.

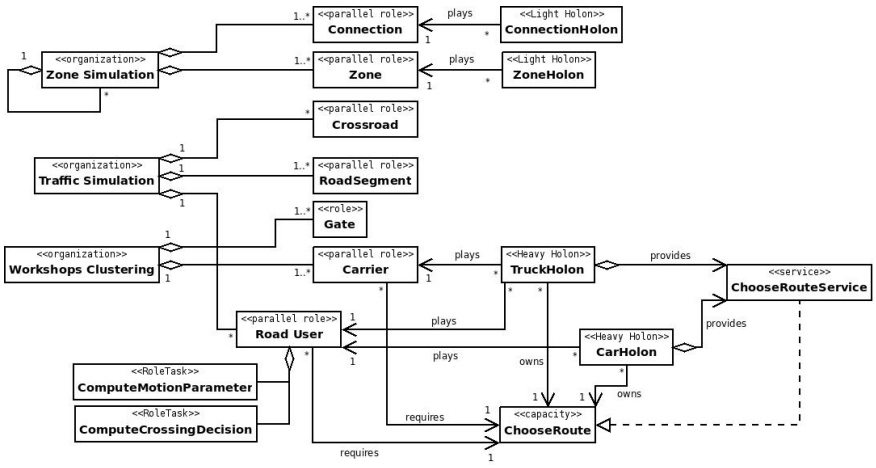

Fig. 21 A fragment of the Holon Architecture designed for the AMP case study

5.2 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 pattern source code may require some adaptation work; for instance, it is often necessary to adapt the interface with the remaining part of the application.

5.3 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 together 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.

5.4 Testing activities

The approach used for tests in ASPECS consists in successively testing each context from the role (the smaller one) to the entire system. Since we use conventional software engineering approaches, these activities will not be extensively discussed here but only a brief description will be provided.

The first activity (organisations and roles unit test) 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. Holon Unit test is the second level of test; it aims at validating the global holon's behaviour. A particular attention is paid to holon dynamics especially to testing rules that govern holon creation, management (task attribution) and the process of members' integration. Each holon is individually tested. The third testing activity, integration test, aims at verifying if the system effectively fulfils the requirements identified in the Domain Requirement Description activity (a great relevance is now given to verification of non-functional requirements that can hardly be tested in the previous test activities).

The ASPECS process also enables the use of formal methods, such as model checking and theorem proving [25], but these aspects are not discussed here due to space concern.

5.5 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 suitable version of *Holon* 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[11], *Janus* provides a full set of tools to manage non-threaded holons execution.

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

5.6 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), their relationships with external devices (sensors, actuators used/accessed by agents) and resources. This activity also details how to perform 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 partition between the various holons used to develop the application. This partition is mainly performed according to localisation of resources 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 new *Janus* kernels at runtime then, the way to deploy organisations and their associated roles, on newly discovered kernels, has to be described too.

Figure 22 illustrates the deployment diagram for the AMP study case. Three physical nodes are considered; each of them is running a *Janus* kernel and is connected to the other nodes via a network connection. The first kernel hosts the 3D engine and the holons playing the Zone role that correspond to the plant. All the other holons are a-priori instantiated on one of the two remaining kernels. Of course the location of the holon on the *Janus* kernel federation could evolve at runtime.

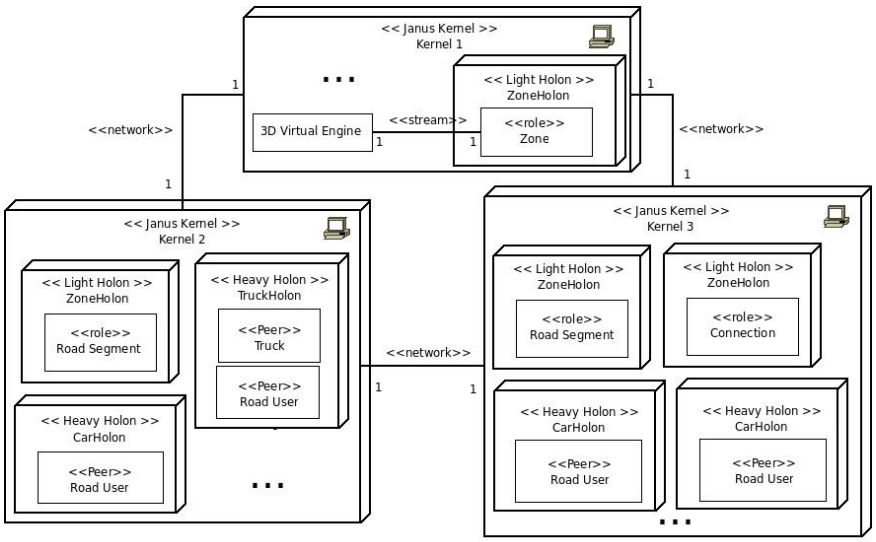

Fig. 22 A fragment of the Deployment Diagram of the AMP case study

6 Comparisons with existing Agent-Oriented Methodologies

This section presents an evaluation and comparison of nine agent-oriented software engineering methodologies. The objective is to emphasise the major similarities and differences between ASPECS and some of the most known existing AOSE methodologies. This study is mainly based on a feature analysis approach and it is inspired by the work of *Tran and Low* [51]. The structure of our study is based on 13 criteria, grouped into three categories. Some of these criteria come from the work of *Tran and Low* [51], while some others have been purposefully introduced to emphasise the evaluation of complex systems related features. These criteria are described in table 3.

	Criteria	**Description**
Process-related	Development life-cycle	What development lifecycle best describes the methodology (e.g., waterfall or iterative)?
	Coverage of the life-cycle	What phases of the lifecycle are covered by the methodology (e.g., analysis, design, and implementation)? Development perspective: What development perspective is supported (i.e., top-down, bottom-up or hybrid)?
	Application domain	Is the methodology applicable to any application domain (i.e., domain independent) or to a specific domain (i.e., domain dependent)?
Model-related	OCMAS or ACMAS	Which modelling perspective is the methodology adopting ? Organisation centered (OCMAS) or Agent-centered (ACMAS)? Is it holonic ?
	System structure	Does the methodology provide means to catch different levels of abstraction to in system conceptualisation ?
	System-Environment interface	Does the methodology provide means to model system delimitation and associated interactions with the outside ?
	Knowledge Model	Does the methodology provide holistic model of the structure of the domain knowledge and the interaction and dependencies of knowledge components in the system ?
	Formal support	Does the methodology have formal foundations ?
Supportive-feature	Standard Integration	Does the methodology respect the main standards that governs AOSE ? Process description: SPEM, UP; Modelling language: UML; Platform: FIPA, MAF;
	Software support	Is the methodology supported by tools and libraries? IDE, platform ?
	Ontology	Does the methodology provide support for the use and specification of ontology in a MAS.
	Open systems	Does the methodology provide support for open systems (i.e., dynamic addition/removal of agents, organisations)?
	Dynamic structure	Does the methodology provide support for dynamic structure (i.e., self-organisation, dynamic reconfiguration of the system)?

Table 3 Comparisons criteria of the evaluation, inspired from [51]

Results of this comparative analysis are summarised in table 4. Target systems of the ASPECS design process are mainly complex open systems; the evaluation is thus done under this perspective. Selected criteria correspond to some of the major points to fulfil in order to model such a type of systems with a specific emphasis on the organisational and holonic perspectives. This evaluation mainly focuses on organisation-centred AOSE design processes. A specific attention is paid to the Anemona design process that is currently the only other methodology explicitly dealing with holonic multi-agent systems.

The remainder of this section describes the main results of the comparative analysis by clustering them on the basis of criteria that distinguish ASPECS from the other existing approaches.

Development life-cycle Just like ASPECS, four of the studied processes adopt a formally described development lifecycle (e.g. PASSI, INGENIAS, ANEMONA, ADELFE) and four of them provide an informal description (e.g. GAIA, ROADMAP,TROPOS, PROMETHEUS). However, most of these methodologies adopt an approach inspired by object-oriented engineering [8] and thus follow a highly iterative development process. TROPOS is the exception because it is founded on the i* modelling framework that focuses on requirements engineering and it is centred on the intentional characteristics of the agent [10].

Concerning the coverage of the life-cycle, GAIA, ROADMAP and TROPOS mainly cover analysis and design phases, while ASPECS covers the entire development process like INGENIAS, ADELFE [4], PROMETHEUS and PASSI.

Application Domain Most of the existing methodologies are domain-independent with the exception of ADELFE and ANEMONA. ADELFE focuses on the development of adaptive multi-agent systems. This type of systems is composed of agents that have a strong relationship with their environment and cooperate with other agents to achieve a specified function.

ANEMONA was the first process to be proposed for designing holonic systems. However, it is domain-dependent and specialised in the field of holonic manufacturing systems. This process is based on the various holon architectures proposed in PROSA [6]. ASPECS is domain-independent and is conceived to be as independent as possible from specific holon architectures (although last activities of the Implementation phase exhibit a necessary link with the adopted implementation platform: *Janus*).

PROSA (the implementation architecture used by ANEMONA) provides four reference holon architectures to be used for the design of holonic manufacturing systems: product, resource, order and staff holon. Besides, PROSA aims at structuring the design of specific system architecture by defining a unified terminology, a generic system structure, the kinds of system components, their responsibilities, design details and models to be drawn. Holon architectures defined in PROSA may be used within the ASPECS design process when they fit designer choices/needs.

Organisational-Centred vs. Agent-Centred MAS In analysis and design of MAS, AOSE methodologies evolved from an initial vision where the system was mainly centred on the agent and its individual aspects (Agent-Centred MultiAgent System), to a vision where the system is now considered as an organisation in which agents form groups and hierarchies, and follow rules and specific behaviours [2] (Organisation-Centred MultiAgent System). The evolution of GAIA (from the first release [53] to the new one [54]) and TROPOS (from [5, 27] to [38]) design processes are probably the most striking examples.

Within organisation-centred approaches, two major trends can be distinguished according to the vision adopted on the concept of organisation [2, 15]: (i) The first is based on the concepts of Role, Group and their relationship, and does not explicitly address the concept of social norm. This approach is adopted by metamodels such as AALAADIN [20] or MOISE [31], and processes like INGENIAS, ANEMONA or TROPOS. (ii) The second focuses more on the concept of norm and explicitly

defines control policies and rules to be established and followed. It is associated with methodologies such as GAIA, SODA [42], OMNI [17], and more generally with the notion of electronic institutions [18].

ASPECS adopts the first perspective and in the set of existing approaches, AALAADIN is probably the one that shares most common points with ASPECS about these organisational concepts. MESSAGE or INGENIAS consider the distinction between Role and Agent as analogous to that between Interface and Object Class. Kristensen and Osterbye [39], studying the notion of role for objects, called such a vision of role as the *Filter Metaphor* and discussed the problems related to such an approach: "*This is mistaken because the filter metaphor implies that the persons*[12] *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 an expected behaviour (a set of tasks ordered by a plan) and a set of rights and obligations in the organisation context (refer to section 2.2 for more details). In the implementation phase, the role exists as a complete entity disposing of its own characteristics and behaviour.

Abstraction levels Complex systems are often a nested network of complex adaptive systems. Indeed, at the very heart of the definition of a complex system we find the notion of emergent behaviour and the possibility of looking at it at different levels of abstraction/observation. In a complex system what we see at one level of abstraction as a whole, might be decomposed in a set of entities at a lower level. Where to set the limit, what to consider as an indivisible component has always been, at least in MAS design, a question of point of view. A design process aimed at modelling such complex systems has to provide a mean for catching the various levels of abstraction. Only few design approaches enable the integration of different abstraction levels in a single model. ROADMAP provides a revised version of the GAIA role model to include various levels of abstraction during the analysis phase and allows an iterative decomposition of the system.

When considering agents as atomic, MAS designers are forced to capture only one of a multitude of possible levels of abstraction. Conversely, we take into consideration multiple levels of abstraction by introducing the concept of holon as a building block, and the description of the holarchy as a structure for composing multi-level holonic organisations.

High-level Features According to our knowledge, ASPECS is the only process that supports both open and dynamic systems and merges an agent-oriented approach with a knowledge-engineering approach based on the prominent role of ontology.

In its first release, GAIA was mainly designed to handle small-scale and closed systems. This choice made it inappropriate for engineering complex open systems [36], GAIA has then been extended by introducing the support for open large sys-

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

tems. However, GAIA still does not consider a holistic model of Domain Knowledge (structure, dependencies between knowledge elements). This forbids sharing system knowledge, reusing, extending and maintaining it in a modular fashion [36]. ROADMAP extends GAIA by introducing a knowledge model for the description of the system domain knowledge. Knowledge components are then assigned to roles. In this sense, we use a similar approach in ASPECS. However, in ASPECS the ontology is considered as a reference point and a source of guidelines for many process activities. Moreover in ASPECS the general description of the problem knowledge is clearly separated from the knowledge that is specific to the solution. In a certain sense, we may consider that in ASPECS we adopt a model driven approach even for system knowledge, this point of view encourages knowledge reusability, maintainability and sharing.

To conclude, the most important features of the ASPECS design process are:

1. its intrinsic ability to catch the various levels of abstraction of a complex system; this occurs during the analysis phase by using an organisational hierarchy and in the design phase by using a holarchy.
2. its aspiration to span the entire software development process from requirements to deployment, and to completely define the process and its various components (since a detailed process description is too huge to be reported in a scientific paper, all the details are reported in the ASPECS website).
3. its ambition to ease the reuse and the extension of domain and application knowledge and models by the use of ontologies; its consideration of the organisation concept as a reusable module independent from the agent architecture (i.e. an organisational design pattern).
4. its integration of multiple viewpoints in the analysis and design of a system by using an organisational perspective throughout the life-cycle and its combination of holonic and classical agent-oriented approaches with knowledge-engineering-based approaches.
5. its explicit modelling of the system-environment relationship by using boundary roles, capacities and resources to provide a holistic description of the system environment. This aspect eases to handle environment changes and thus facilitates the deployment in dynamic and heterogeneous environments.

7 Conclusions and future works

This paper presents the ASPECS software development process also with the help of a concrete case study from 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 those exhibiting a hierarchical structure.

The respect and integration of the most diffused AOSE domain standard specifications is one of the basis of our approach. The description of the development process is thus based on SPEM (reported on the website), graphical notations are based on UML, and FIPA standards are also largely adopted. ASPECS notation extends UML especially to take into account organisational and holonic concepts.

	Criteria	PASSI	INGENIAS	ANEMONA	GAIA	ROADMAP	TROPOS	PROMETHEUS	ADELFE	ASPECS
Process-related	Development lifecycle	Iterative across and within all phases	Unified software development process (UP)	Unified software development process (UP)	Iterative within each phase but sequential between phases	Iterative within each phase but sequential between phases	Iterative and incremental	Iterative across all phases	Rational Unified Process (UP)	Iterative across and within all phases
	Coverage of lifecycle	Analysis, Design, Implementation and Deployment	Analysis, Design, Implementation and Deployment	Analysis, Design, Implementation	Analysis and Design	Analysis and Design	Analysis and Design	Analysis and Design, Implementation	Analysis, Design, Implementation	Analysis, Design, Implementation and Deployment
	Application Domain	Independent	Independent	Dependent - Holonic manufacturing systems	Independent	Independent	Independent	Independent	Dependent - adaptive systems	Independent
Model-related	OCMAS vs. ACMAS	ACMAS but Role-oriented analysis	OCMAS (structure and implicitly norms)	OCMAS (structure and implicitly social norms), holonic	OCMAS (social norms and structure)	OCMAS (structure)	OCMAS (structure)	ACMAS	ACMAS	OCMAS (structure and implicitly social norms), holonic
	Number of abstraction levels	1	1	n (agent composition)	1	n (role hierarchy)	1	1	2 (local and global)	n (organisational hierarchy and agent composition)
	System-Environment Interface	Yes, Resource modelling	Yes, environment model	Yes, environment model	Implicit, (sensors and effectors)	Yes, Environment Model	No	Yes, Percepts and Actions descriptor	Yes, detailed architecture document	Yes, IRI (boundary roles) and ODD models
	Knowledge Model	Yes	Yes	Yes	No	Yes	No	No	No	Yes
	Formal foundations	No	No	No	No	No	Yes, i* and formal tropos	No	No	Partially, OZS
Supportive-Feature	Standards Integration	SPEM, UML, FIPA	UP, SPEM, MDD, MOF, UML, FIPA	UP, SPEM, FIPA, UML	SPEM[a]	-	-	-	UP, SPEM, UML, FIPA	SPEM, UML, FIPA
	Tools : IDE	Metameth and PTK	IDK	-	-	Rebel	T-Tool	PDT	OpenTool	Planned
	Tools : Platforms and libraries	Guidelines, tool for JADE and FIPA-OS	Tools for JADE, BDI Agents	Guidelines for JADE, PROSA	Guidelines for JADE	Guidelines for JADE	-	JACK	-	JANUS
	Ontology	Yes	No	No	No	No	No	No	No	Yes
	Open Systems	No	No	No	Yes	Yes	No	No	Yes	Yes
	Dynamic Structure	No	No	No	No	No	No	No	No	Yes

Table 4 Comparisons between ASPECS and nine well-known AOSE design processes

[a] not done by primary authors

ASPECS allows the modelling of a system with 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 simple enough 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 makes explicit (by means of boundary roles) 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 interface between the environment and the system eases the deployment of applications on dynamic and heterogeneous environments.

Domain knowledge is explicitly encoded in the Problem and Solution ontologies. ASPECS thus presents a holistic model of the structure of the domain knowledge as well as the interactions 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.

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.

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 used to implement our holonic applications. The second is Janeiro, a CASE tool that deals with the analysis and design aspects.

Further works will particularly focus on the integration of formal notations and methods especially OZS. OZS (Object-Z and Statechart [29]) 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).

Acknowledgements Authors would like to thank 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. *Software Engineering Body of Knowledge*. IEEE Computer Society, 2004.

2. E. Argente, V. Julian, and V. Botti. Multi-Agent System Development Based on Organizations. In *CoOrg'06*, volume 150 of *Electronic Notes in Theoretical Computer Science*, pages 55–71. Elsevier, May 2006.
3. C. Bernon, M. Cossentino, and J. Pavón. An overview of current trends in european aose research. *Informatica*, 29(4):379–390, July 2005.
4. C. Bernon, M.-P. Gleizes, S. Peyruqueou, and G. Picard. ADELFE, a methodology for adaptive multi-agent systems engineering. In *ESAW*, volume 2577 of *LNAI*, pages 156–169, Madrid, Spain, September 2002. Springer-Verlag.
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, May 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, *AOSE 2001*, volume 2222 of *LNCS*, pages 119–135. Springer Verlag, 2002.
8. L. Cernuzzi, M. Cossentino, and F. Zambonelli. Process models for agent-based development. *Journal of Engineering Applications of Artificial Intelligence (EAAI)*, 18(2), March 2005.
9. 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.
10. L. Chung, B. A. Nixon, and E. S. K. Yu. Dealing with change: An approach using non-functional requirements. *Requirements Engineering*, 1(4):238–260, 1997.
11. A. Cockburn. Structuring use cases with goals. *Journal of Object-Oriented Programming*, pages 56–62, Nov/Dec 1997.
12. A. Cockburn. *Writing Effective Use Cases*. Addison-Wesley, 2000.
13. 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.
14. 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*, 1(1):91–121, April 2007.
15. L. d. R. Coutinho, J. S. a. Sichman, and O. Boissier. Modeling organization in MAS: a comparison of models. In *SEAS*, Uberlândia, October 2005.
16. R. A. Dahl. *Polyarchy: Participation and Opposition*. Yale University Press, New Haven, 1971.
17. M. Dignum, J. Vazquez-Salceda, and F. Dignum. OMNI: Introducing Social Structure, Norms and Ontologies into Agent Organizations. In *PROMAS@AAMAS*, volume 3346 of *LNAI*, pages 181–198. Springer, July 2005.
18. M. Esteva, J. A. Rodríguez-Aguilar, C. Sierra, P. Garcia, and J. L. Arcos. On the formal specifications of electronic institutions. In *Agent Mediated Elec-*

tronic Commerce, The European AgentLink Perspective, pages 126–147, London, UK, 2001. Springer-Verlag.
19. J. Ferber. *Multi-Agent Systems. An Introduction to Distributed Artificial Intelligence*. Addison Wesley, London, 1999.
20. J. Ferber, O. Gutknecht, and F. Michel. From Agents to Organizations: an Organizational View of Multi-Agent Systems. In *AOSE-IV@AAMAS03*, volume 2935 of *LNCS*, pages 214–230. Springer Verlag, mar 2004.
21. Foundation For Intelligent Physical Agents. *FIPA RDF Content Language Specification*, 2001. Experimental, XC00011B.
22. Foundation For Intelligent Physical Agents. *FIPA ACL Message Structure Specification*, 2002. Standard, SC00061G.
23. Foundation For Intelligent Physical Agents. *FIPA Communicative Act Library Specification*, 2002. Standard, SC00037J.
24. N. Gaud, S. Galland, V. Hilaire, and A. Koukam. An Organisational Platform for Holonic and Multiagent Systems. In *PROMAS-6@AAMAS'08*, Estoril, Portugal, May 12-16th 2008.
25. N. Gaud, V. Hilaire, S. Galland, A. Koukam, and M. Cossentino. A verification by abstraction framework for organizational multi-agent systems. In *AT2AI-6@AAMAS'08*, Estoril, Portugal, May 2008.
26. C. Gerber, J. Siekmann, and G. Vierke. Holonic multi-agent systems. Technical Report DFKI-RR-99-03, DFKI - GmbH, 1999.
27. F. Giunchiglia, J. Mylopoulos, and A. Perini. The Tropos Software Development Methodology: Processes, Models and Diagrams. Technical Report 0111-20, ITC - IRST, 2002. Submitted AAMAS Conference 2002. A Knowledge Level Software Engineering 15.
28. T. Gruber. Toward principles for the design of ontologies used for knowledge sharing. *International Journal Human-Computer Studies*, 43(Issues 5-6):907–928, November 1995.
29. 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.
30. O. Gutknecht and J. Ferber. Madkit: a generic multi-agent platform. autonomous agents. In *AGENTS 2000*, pages 78–79, Barcelona, 2000. ACM Press.
31. M. Hannoun, O. Boissier, J. S. Sichman, and C. Sayettat. MOISE: An Organizational Model for Multi-agent Systems. In M. Monard and J. Sichman, editors, *Advances in Artificial Intelligence, IBERAMIA-SBIA*, pages 156–165, Brazil, 2000.
32. B. Henderson-Sellers. Method engineering for OO systems development. *Commun. ACM*, 46(10):73–78, 2003.
33. 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, *ESAW*, number 1972 in LNAI. Springer Verlag, 2000.
34. C. Iglesias, M. Garijo, J. Gonzalez, and J. Velasco. *Analysis and design of multi-agent systems using MAS-CommonKADS*, volume 1365 of *LNAI*, chapter Intelligent agents IV: Agent theories, architectures, and languages, pages

313–326. Springer-Verlag, 1998.
35. N. Jennings. An agent-based approach for building complex software systems. *Commun. ACM*, 44(4):35–41, April 2001.
36. T. Juan, A. Pearce, and L. Sterling. ROADMAP: Extending the GAIA methodology for complex open systems, 2002.
37. A. Koestler. *The Ghost in the Machine*. Hutchinson, 1967.
38. M. Kolp, P. Giorgini, and J. Mylopoulos. Multi-agent architectures as organizational structures. *Autonomous Agents and Multi-Agent Systems*, 13(1):3–25, 2006.
39. B. Kristensen and K. Osterbye. Roles: Conceptual abstraction theory and practical language issues. *Theory and Practice of Object Systems*, 2(3):143–160, 1996.
40. Object Management Group. *MDA Guide, v1.0.1, OMG/2003-06-01*, June 2003.
41. J. Odell, M. Nodine, and R. Levy. A metamodel for agents, roles, and groups. In J. Odell, P. Giorgini, and J. Müller, editors, *AOSE*, LNCS. Springer, 2005.
42. A. Omicini. SODA: Societies and Infrastructures in the Analysis and Design of Agent-Based Systems. In Springer-Verlag, editor, *AOSE*, volume 1957 of *LNCS*, pages 185–193, 2000.
43. L. Padgham and M. Winikoff. Prometheus: A methodology for developing intelligent agents. In *AOSE*, July 2002.
44. J. Pavón, J. Gómez-Sanz, and R. Fuentes. The INGENIAS methodology and tools. In *Agent-Oriented Methodologies*, pages 236–276. Idea Group Publishing, NY, USA, June 2005.
45. S. Sauvage. Agent oriented design patterns: A case study. In *AAMAS '04*, pages 1496–1497, Washington, DC, USA, 2004. IEEE Computer Society.
46. J. Searle. *Speech Acts*. Cambridge University Press, Cambridge, UK, 1969.
47. 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.
48. H. A. Simon. *The Science of Artificial*. MIT Press, Cambridge, Massachusetts, 3rd edition, 1996.
49. I. Sommerville. *Software Engineering*. International Computer Science Series. Addison Wesley, Pearson Education, seventh edition edition, 2004.
50. SPEM. *Software Process Engineering Metamodel Specification, v2.0, Final Adopted Specification, ptc/07-03-03*. Object Management Group, March 2007.
51. Q.-N. N. Tran and G. C. Low. *Agent-Oriented Methodologies*, chapter XII: Comparison of Ten Agent-Oriented Methodologies, pages 341–367. Idea Group, 2005.
52. K. Wilber. *Sex, Ecology, Spirituality*. Shambhala, 1995.
53. M. Wooldridge, N. R. Jennings, and D. Kinny. The GAIA methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.
54. F. Zambonelli, N. Jennings, and M. Wooldridge. Developing multiagent systems: the GAIA methodology. *ACM Trans. on Software Engineering and*

Methodology, 12(3), 2003.