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

Massimo Cossentino Nicolas Gaud Vincent Hilaire
Stéphane Galland Abderrafiâa Koukam

November 22, 2007

Multiagent Systems Group,
System and Transport Laboratory,
University of Technology of Belfort Montbéliard,
90010 Belfort cedex, France.
http://set.utbm.fr

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

Abstract

Holonic multiagent systems (HMAS) offers a promising software engineering approach for developing complex open software systems. However the process of building MultiAgent Systems (MAS) and HMAS is mostly different from the process of building more traditional software systems and it introduces new design and development challenges. This paper introduces an agent-oriented software process for engineering complex systems called ASPECS. ASPECS is based on a holonic organisational metamodel and provides a step-by-step guide from requirements to code allowing the modelling of a system at different levels of details using a suite of refinement methods.
Keywords : 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. Increasing both in size and complexity, actual software systems are expected to be distributed, open and highly dynamic. Multiagent systems are emerging as probably one of the most adopted software engineering paradigm for developing complex software systems [17, 31]. However the current practice of Multi-Agent System (MAS) design tends to be limited to individual agents and small face-to-face groups of agents that operate in closed systems [20]. This practice seems in contradiction with the current evolution of software system requirements and what previously reported about complex systems.

According to Simon [26], complex systems often (if not always) exhibit a hierarchical configuration[1]. And the idea that the architecture of a complex system can be explained and understood using a "hierarchical organisation structures" is shared by a relevant number of scientists [7, 30]. Several metamodels and methodologies have been proposed for MAS [1]. However, most of them see agents as atomic entities. Considering agents as composed entities thus enabling a modelling of nested hierarchies may offer a more adapted and manageable way to model complex systems.

Giving this landscape, we advocate the use of holonic multiagent systems (HMAS) combined with an organisational approach for developing complex software systems. This paper introduces an agent-oriented software process for engineering complex systems called ASPECS. The process can be considered as an evolution of the PASSI [4] methodology for modelling HMAS and it also collects experiences about holon design coming from the RIO approach [15]. The construction of the new process has been performed according to the situational method engineering paradigm [14, 24] and the approach described in [5]. The complete description of the method adopted for building the ASPECS process is out of the scope of this paper. It is sufficient to say that the definition of the MAS metamodel adopted by the new process has been the first step and from this element all the others (activities, guidelines, workflow) have been deducted [5]. For this reason the description of each phase of ASPECS will start from the portion of MAS metamodel instantiated/refined by the activities.

ASPECS is based on a holonic organisational metamodel and provides a step-by-step guide from requirements to code allowing the modelling of a system at different levels of details using a suite of refinement methods. Using a holonic perspective, the designer can model a system with entities of different granularities. He can recursively model subcomponents of a bigger system until he achieves a stage where 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 and a hierarchical structure composed of holons is called a *holarchy*. A holon can be seen, depending on the level of observation, either as an autonomous "atomic" entity or as an organisation of holons (this is often called the *Janus effect* [18]). Holonic Systems have been already applied to a wide range of applications. Thus it is not surprising that a number of models and frameworks have been proposed for these systems, for instances PROSA [2] and MetaMorph [19, 25]. 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 [12]. Number of methodologies and metamodels with a clear organisational vision have been already proposed: metamodels like AGR [8], RIO [15], and methodologies like GAIA [31], INGENIAS [22], MESSAGE [3], and SODA [21]. 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 [16, 31]. As pointed out by Ferber [8, 9], 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 architec-

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

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

The paper is organised as follows: after a quick introduction to the proposed methodology (section 2), the complete ASPECS software process us presented (sections 3–5). Finally, conclusions are drawn in section 6.

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. The target scope for the proposed approach can be found in complex systems and especially hierarchical complex systems. The main vocation of ASPECS is towards the development of societies (organisations) of holonic (as well as not-holonic) multiagent systems.

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

The ASPECS process structure (in terms of process metamodel) is based on the Software Process Engineering Metamodel Specification (SPEM) [28] proposed by the OMG. According to this metamodel, the software process of ASPECS is based on three main levels: *Phases*, *Activities* and *Tasks*. A Phase delivers a composite work product (composed of one or more documents that can belong to different work product types), a phase is composed of a number of activities that are in turn decomposable into tasks. An Activity delivers a main work product (like a diagram or a text document) and it is composed of a number of Tasks. A task contributes to the production of a work product (usually by delivering a part of it), and it instantiates/relates/refines MAS metamodel elements.

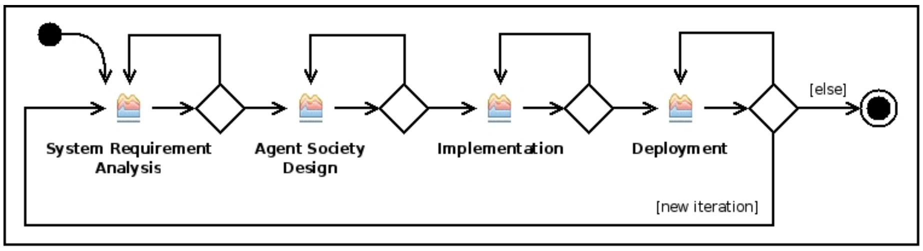

Figure 1: Life Cycle of ASPECS

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

1. System Requirements Analysis: the organisational description of the problem and the ontological description of the application's context.

2. Agent Society Design: the design of the agent society that is able to provide a solution to the previously described problem, and a refinement of the application context description.

3. Implementation: the description of the holon architectures, associated code production and tests.

4. Deployment: the description of the application deployment.

Because of space concerns, each activity will be briefly described in the following sections but the interested reader may refer to the following address for a complete description : http://set.utbm.fr/index.php?pge=352&lang=fr.

In each activity, the related concepts of the metamodel are described. By definition a metamodel is a “model of a model”, and it provides an explicit representation of the constructs and relationships needed to build specific models within a domain of interest. As it happens for the PASSI MAS meta-model, the ASPECS one introduces three domains. The first is the *problem domain* dedicated to the description of a problem independently of a specific solution and it is used in the first phase dedicated to System Requirement Analysis, that is described in section 3 (see figure 2). The second is the *agency domain* which introduces agent concepts to describe an agent solution on the basis of the elements of the problem domain, and it is related to the design of the Agent Society (see section 4, figure 4). The *solution domain* is the third and last one; it includes the elements used to implement at the code level the solution described in the agency domain. The implementation phase is based on the concepts introduced in this last domain and is detailed in section 5 (see figure 8). A complete description of the ASPECS metamodel can be found in [6].

3 System Requirement Analysis

System requirements analysis phase aims at providing a full description of the problem based on the concepts defined in the *Problem Domain* area of the metamodel. This last is presented in figure 2. Each concept of the problem domain will be detailed in the activity where it is defined. The various activities involved in the System Requirement phases and the set of associated diagrams are described in figure 3. In the following subsections a description of each activity is presented.

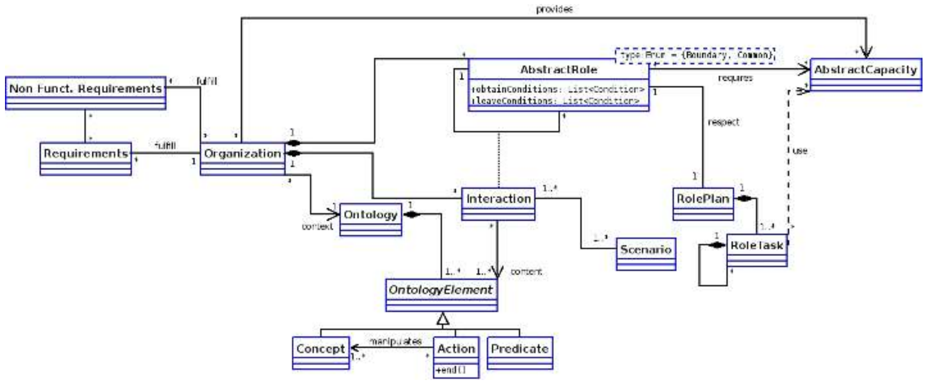

Figure 2: The UML diagram of the ASPECS metamodel Problem Domain

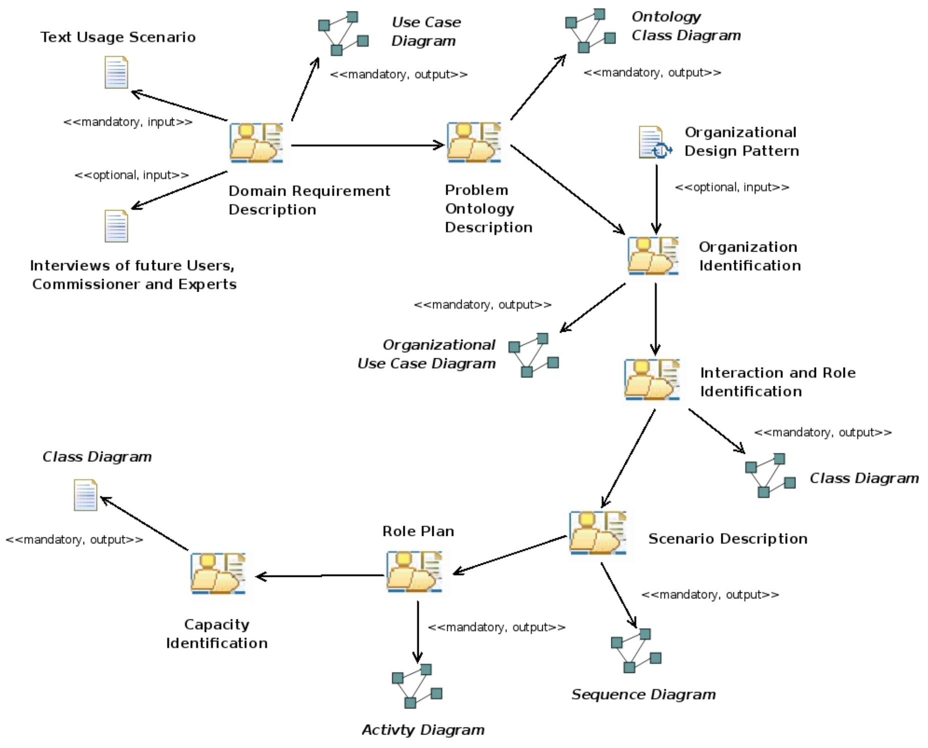

Figure 3: Details of the System Requirements Analysis Phase

Domain Requirements Description (DRD) : provides an overview of the system's functionalities. This activity aims at gathering needs and expectations of application stakeholders and at providing a complete description of the behaviour of the application to be developed. In the proposed approach, these requirements (both functional and non functional) should be described using the specific language of the application domain and a user perspective.

Problem Ontology Description (POD) : provides an overview of the problem domain. Stakeholders naturally express requirements in their own terms and with implicit knowledge of their own works [27]. Therefore the aim of this activity is deepening the understanding of the problem by complementing the usual requirements description 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.

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

Interactions and Role Identification (IRI) : aims at decomposing a global behaviour embodied by an organisation into smaller interacting behaviours. Each of these finer grained behaviours will be represented by an AbstractRole. Thus, an AbstractRole is the abstraction of a behaviour in a certain context defined by the organisation and confers a status within this context. AbstractRoles interact according to one or more specific interaction patterns.

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

Scenario Description (SD) : describes the sequence of interactions among the roles involved in each scenario. A Scenario describes how roles interact and coordinate to satisfy goals assigned to their organisation. Scenarios description occurs just after *OID* and *IRI* activities and at this stage it is possible to assign to each requirement an organisation and a set of interacting behaviours (enacted by involved roles). The challenge is now describing how these different roles interact to realise the scenario.

Role Plan (RP) : conceive for each role a plan that could fulfil the part of the organisation requirements that have been delegated to the role under study. In this context a plan describes how a goal can be achieved. It is the description of how to combine and order interactions, external events, and RoleTasks to fulfil a (part of a) requirement (the goal). A RoleTask is the specification of a parameterised behaviour in form of a coordinated sequence of subordinate units (a RoleTask can be composed of other RoleTasks). The definition of these units can be based on capacities, required by the role. At this step, we focus on each role in order to define a plan that may accomplish the requirements coming from the previous parts of the design. The first task in this activity consists in detailing responsibilities assigned to the currently designed role. Then for each of them, a set of RoleTasks has to be identified for accomplishing the assigned requirements. The final step consists in determining transitions between the various activities and the set of associated conditions. In a second iteration each task will be examined to be eventually decomposed in order to determine if it requires something external to the role. If this is the case then a new capacity will be created and the role will refer to it.

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

In other words, the main objective of the Capacity Identification activity (CI) is the definition of generic role behaviours by identifying which know-how a role requires from the individual that will play it; The capacity element constitutes a layer between the role behaviour and the future entities which will want to play this role. Basing the description of role behaviour on capacities gives to the role more genericity and modularity. A complete description of the concepts of capacity and template that describes a capacity can be found in [23]. For the scope of this paper it is worth to say that the capacity is a description of what an organisation (and therefore one of its composing roles) is able to do without any kind of specification on how to do it. This means that the results prescribed by a capacity may be reached by adopting different strategies (the realisation of the capacity is a concern of the Agency Domain and will be discussed later).

4 Agent Society Design

This phase aims at designing a society of agents, whose global behaviour is able to provide an effective solution to the problem described in the previous phase and to satisfy associated requirements. After modelling the problem in terms of organisations, roles, capacities and interactions, the objective is, now, to provide a model of the agent society involved in the solution in terms of social interactions and dependencies among entities (Holons and/or Agents).

The Agency Domain part of the HMAS metamodel is reported in Figure 4; some of its elements (Group, AgentRole, Capacity, AgentTask) are a specialisation of other elements defined in the Problem Domain (the two domains are separated by a dashed line); they constitute the backbone of our approach and are refined from one domain to the other in order to contribute to the final implementation of the system.

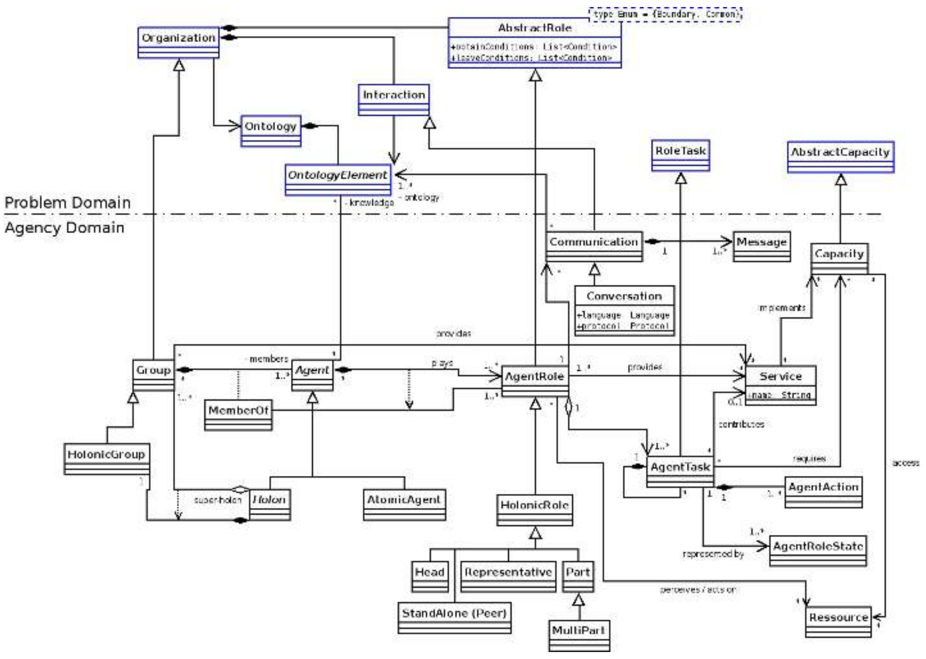

Figure 4: The UML diagram of the ASPECS metamodel

Details of activities and the set of associated diagrams involved in the Agent Society

Design phase are described in figure 5. In the following sub-sections each activity will be detailed.

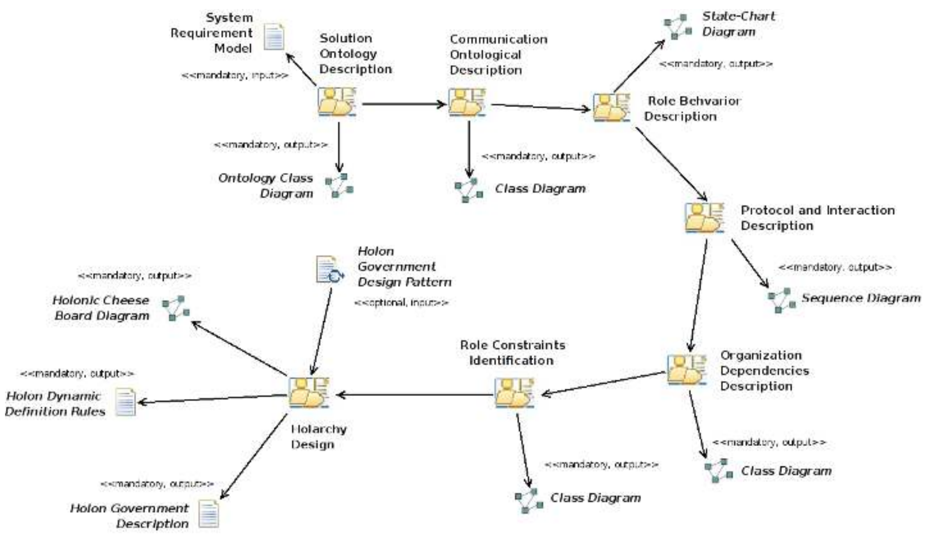

Figure 5: The Agent Society Design Phase

Solution Ontology Description (SOD) : consists in refining the problem ontology described during POD by adding new concepts related to the agent-based solution and refining the existing ones. Concepts, predicates and actions of this ontology are now also intended to be used for describing information exchanged during communications among roles. The first task consists in refining existing concept descriptions and identifying new ones (related to the agency-level solution that is under development). Actions and predicates should also be added.

Communication Ontological Description (COD) : describes communications and conversations among roles. Communications are a more refined way for interacting compared to the basic Interactions allowed to the AbstractRole. Interactions identified in the problem domain are specialised in this domain in communications. The model of role communication is based on the assumption that two roles wishing to interact, share a common ontology. This common knowledge is represented in the communication by a set of Ontology Elements. A conversation is a specialisation of a communication in which content (Language, Encoding, Ontology) and control (Protocol) of the communication are detailed. A conversation mainly consists of FIPA[2] speech acts [10, 11] and protocols. A protocol defines the sequence of expected messages.

Role Behaviour Description (RBD) : definies the complete life-cycle of a role by integrating previously identified RoleTasks, Abstract Capacities, comunications/-conversations in which it is involved and the set of (or part of) requirements it has to fulfil. AbstractRoles identified during the IRI activity are specialised here in AgentRoles. An AgentRole interacts with the others using communications. The behaviour of the role is described by a set of AgentTasks and AgentActions.

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

AgentTask is the specialisation of the RoleTask. It is aggregated in AgentRole and contributes to provide (a portion of) an AgentRole's service. At this level of abstraction, this kind of task is no more considered atomic but it can be decomposed in finer grained AgentActions. An AgentAction is now the atomic unit of behaviour specification. An action takes a set of inputs and converts them into a set of outputs, though either or both sets may be empty. An example of the most basic AgentAction consists in invoking a capacity or requesting a service (as explained in further subsections).

Role Behaviour Description is a refinement at the Agent Society level of details of the results produced by the Role Plan activity during 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 the intrinsic state-based nature of roles. 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.

Protocol Description (PD) : defines purpose-specific interaction protocols whose need may arise when the description of communications done during the COD (Communication Ontology Description) and SD (Scenario Description) activities do not match any of the existing FIPA protocols. The designer starts from scenarios and the ontological description of communications in order to find the need for a new specific interaction protocol. If this is the case, then he can proceed to the definition of a new protocol that is compliant with the interactions described in scenarios and the communication semantics. It is advisable to refer to the FIPA Interaction protocols library[3] 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.

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

A service implements a capacity thus accomplishing a set of functionalities on behalf of its owner, a role (definition inspired from the Web Services Architecture [29] of the W3C[4]). These functionalities can be effectively implemented by a set of capacities required by the owner role. A role can thus publish several of its capacities and other members of the group can take profit of them by means

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

of a service exchange. Similarly a group, that is able to provide a collective capacity can share it with other groups by providing a service.

The relationship between capacity and service is thus crucial in our metamodel. A capacity is an internal aspect of an organisation or an agent, while the service is designed to be shared among various organisation or entities. A service is created to publish a capacity and thus to allow other entities to benefit from it.

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

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

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

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

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

Holarchy Design (HD) : Two fundamental elements of the MAS metamodel are newly introduced in this activity; they are Agent and Holon. An Agent is an entity which can play a set of roles defined within various organisations; these roles interact each other in the specific context provided by the agent itself. Several AgentRoles are usually grouped in one Agent. The Agent context is given by its knowledge, its capacities and its environment. Roles share this context by the simple fact of being part of the same agent. This mean for instance that

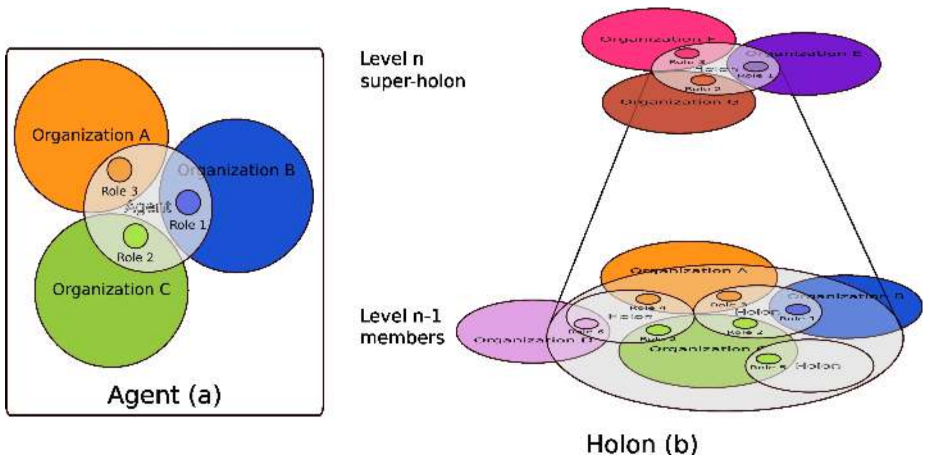

Figure 6: Agent and Holon symbolic representation

an agent can play the role of *Buyer* in an organisation and later the same agent can sell the goods it had just acquired thus playing for the same organisation a different role (*Seller*); conversely, the same agent can also play roles belonging to another organisation (for instance devoted to monitor businesses), i.e. it can play the role (*AffairMonitor*) to trace the results and the performance exploited during the first acquisition process. It is worth to note that the agent is still not an implementation-level element (platform-dependent) but rather it needs further refinements. Figure 6(a) depicts the context defined by an agent as an interaction space for the roles it plays. These roles, in turn, belong to different organisations, each one defining a different context. The concept of Holon is specialised from the Agent one. A holon is thus a set of roles that can be defined on various organisations interacting in the specific context provided by the agent. A holon can play several roles in different organisations and it may be composed of other holons. A composed holon (super-holon) contains at least a single instance of a holonic organisation to precise how members organise and manage the super-holon and a set (at least one) of production organisations describing how members interact and coordinate their actions to fulfil the super-holon tasks and objectives. An atomic (non composed) holon is an AtomicAgent. Figure 6(b) illustrates this definition of holon.

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

a. firstly, the agentification task consists in identifying all the required agents/holons and associating them with the set of roles they have to play. To achieve that, each previously identified organisation is specialised into a Group. This element is used to model groups of Agents that cooperate in order to achieve a goal.

b. The second task focuses on composed holons and aims at identifying a government type for each of them. The objective consists in describing the various rules used to take decisions inside each super-holon.

c. Then, all the previously described elements are merged in order to obtain the complete set of holons (composed or not) involved in the solution. In this way, the complete holarchy of the system is described.

d. The description obtained with the previous tasks is just the initial structure of the system, the last objective is now to specify rules governing holons' dynamics in the system (creation, new member integration, specific self-organisation mechanisms, scheduling policies for roles) in order to support a dynamic evolution of the system holarchy.

Describing an holon internal structure and dynamic is thus a complex task. Three main aspects may be distinguished: Holon Member's modelling, Holon Government, Holon Dynamics.

5 Implementation

This section will give a brief overview of the implementation phase. The details of its activities and the set of associated diagrams are described in figure 7. Concepts at the basis of our implementation, and constituting the Solution Domain part of the HMAS meta-model are reported in Figure 8.

The Implementation phase aims at implementing the agent-oriented solution designed in the previous phase by adapting it to the chosen object-oriented implementation platform. This part of the work is thus dependent on the implementation and deployment platform.

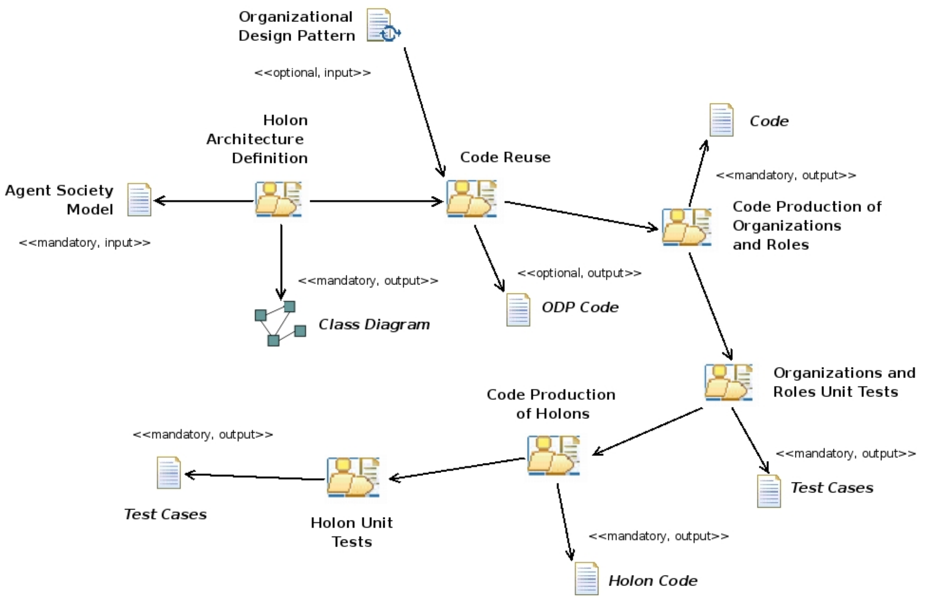

Figure 7: Implementation Phase

A platform called *Janus*[5] 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.

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

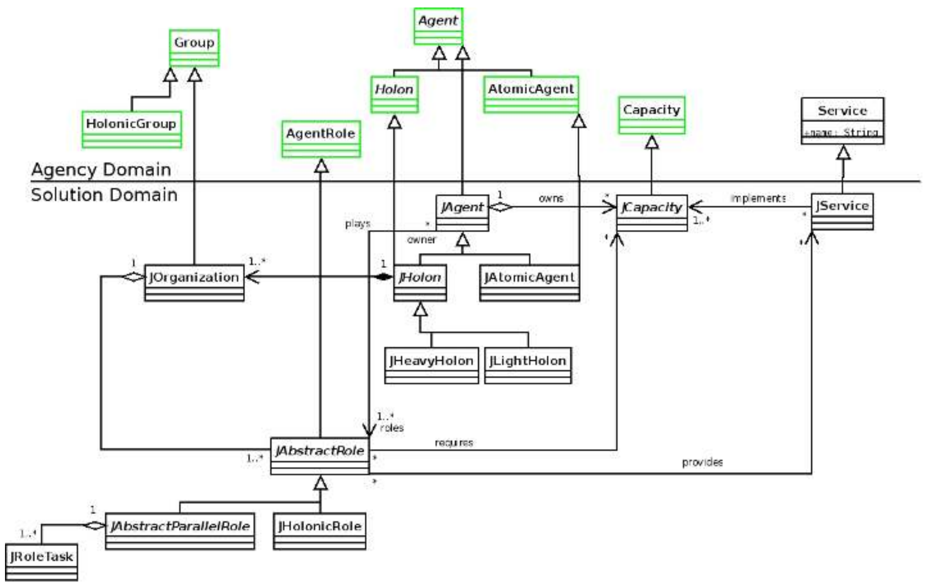

Figure 8: The UML diagram of the Solution Domain of the ASPECS metamodel

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 [9], JADE, and FIPA-OS, in *Janus* the concept of Role is considered as a first class entity. It thus allows to directly implement organisational models without making any assumptions on the architecture of the holons that will play the role(s) of a organisation. An organisation is defined by a set of roles and a set of constraints to instantiate these roles (e.g. maximal number of authorised instances). Thus, organisations designed for an application can be easily reused for another. *Janus* promotes reusability and modularity, moreover the use of organisational design patterns is strongly encouraged. Each organisation is a singleton and it can be instantiated by several groups. Group is the runtime context of interaction. It contains a set of roles and a set of Holons playing at least one of these roles. In addition to its characteristics and its personal knowledge, each agent/holon has mechanisms to manage the scheduling of its own roles. It can change dynamically its roles during the execution of the application (leave a role and request a new one). The life-cycle of a *Janus* agent is composed of three main phases: activation, life, termination. The life of an agent consists in consecutively executing its set of roles and capacities. To describe the personal competences of each agent/holon, *Janus* implements the concept of *JCapacity* that is an abstract description of a competence; each agent can be natively equipped with an implementation of a JCapacity or can dynamically acquire it (this function is still under development). In addition to the integration of these personal characteristics, a holon provides an execution context for roles and capacities.

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

6 Conclusion and perspectives

This paper presented the ASPECS software development process. ASPECS covers the entire software engineering process and it is designed for the development of complex software systems, especially (but not exclusively) those exhibiting a hierarchical structure.

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

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

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

Concerning the environment that is an essential part of MAS, ASPECS through the use of boundary roles explicits the representation of interactions between the system and the necessary environmental entities without making any assumptions on the concrete environment structure. The use of specific capacities as an the interface between the environment and the system renders easy the deployment of applications on dynamic and heterogeneous environments.

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

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

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

Further works will particularly focus on the integration of formal notations and methods especially OZS. OZS (Object-Z and State-chart [13]) 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 state-chart 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 state-charts to Java code).

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

References

[1] C. Bernon, M. Cossentino, and J. Pavón. An overview of current trends in european aose research. *Informatica*, 29(4):379–390, July 2005.

[2] 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.

[3] G. Caire, W. Coulier, F. J. Garijo, J. Gomez, J. Pavón, F. Leal, P. Chainho, P. E. Kearney, J. Stark, R. Evans, and P. Massonet. Agent oriented analysis using Message/UML. In M. Wooldridge, G. Weiß, and P. Ciancarini, editors, *Agent-Oriented Software Engineering II, Second International Workshop, AOSE 2001, Montreal, Canada*, volume 2222 of *LNCS*, pages 119–135. Springer Verlag, 2002.

[4] 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.

[5] M. Cossentino, S. Gaglio, A. Garro, and V. Seidita. Method fragments for agent design methodologies: from standardization to research. *International Journal on Agent Oriented Software Engineering (IJAOSE)*, 1(1):91–121, April 2007.

[6] M. Cossentino, N. Gaud, S. Galland, V. Hilaire, and A. Koukam. A holonic metamodel for agent-oriented analysis and design. In V. Marik, V. Vyatkin, and A. Colombo, editors, *Proc. of the 3rd International Conference on Industrial Applications of Holonic and Multi-Agent Systems (HoloMAS'07)*, volume 4659 of *Springer-Verlag LNAI*, pages 237–246, Regensburg, Germany, September 2007.

[7] M. Edwards. A brief history of holons, October 2003. available at http://www.integralworld.net/edwards13x.html.

[8] J. Ferber and O. Gutknecht. A meta-model for the analysis and design of organizations in multi-agent systems. In *ICMAS'98*, pages 128–135, 1998.

[9] J. Ferber, O. Gutknecht, and F. Michel. From agents to organizations: an organizational view of multi-agent systems. In *Agent-Oriented Software Engineering IV 4th International Workshop*, volume 2935 of *LNCS*, pages 214–230, Melbourne, Australia, mar 2004. Springer Verlag.

[10] Foundation For Intelligent Physical Agents. *FIPA ACL Message Structure Specification*, 2002. Standard, SC00061G.

[11] Foundation For Intelligent Physical Agents. *FIPA Communicative Act Library Specification*, 2002. Standard, SC00037J.

[12] L. Gasser. Mas infrastructure: Definitions, needs and prospects. In *Revised Papers from the International Workshop on Infrastructure for Multi-Agent Systems*, pages 1–11, London, UK, 2001. Springer-Verlag.

[13] 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.

[14] B. Henderson-Sellers. Method engineering for OO systems development. *Commun. ACM*, 46(10):73–78, 2003.

[15] V. Hilaire, A. Koukam, P. Gruer, and J.-P. Müller. Formal specification and prototyping of multi-agent systems. In A. Omicini, R. Tolksdorf, and F. Zambonelli, editors, *Engineering Societies in the Agents' World*, number 1972 in LNAI. Springer Verlag, 2000.

[16] N. Jennings. On agent-based software engineering. *Artificial Intelligence*, 177(2):277–296, 2000.

[17] N. Jennings. An agent-based approach for building complex software systems. *Commun, ACM*, 44(4):35–41, April 2001.

[18] A. Koestler. *The Ghost in the Machine*. Hutchinson, 1967.

[19] F. Maturana. *MetaMorph: an adaptive multi-agent architecture for advanced manufacturing systems*. PhD thesis, The University of Calgary, 1997.

[20] J. Odell, M. Nodine, and R. Levy. A metamodel for agents, roles, and groups. In J. Odell, P. Giorgini, and J. Müller, editors, *Agent-Oriented Software Engineering (AOSE) IV*, LNCS. Springer, 2005.

[21] A. Omicini. SODA: Societies and infrastructures in the analysis and design of agent-based systems. In Springer-Verlag, editor, *Agent-Oriented Software Engineering: First International Workshop, AOSE*, volume 1957 of *LNCS*, pages 185–193, 2000.

[22] J. Pavón, J. Gómez-Sanz, and R. Fuentes. The ingenias methodology and tools. In B. Henderson-Sellers and P. Giorgini, editors, *Agent-Oriented Methodologies*, volume ISBN1-59140-581-5, pages 236–276. Idea Group Publishing, NY, USA, juin 2005.

[23] S. Rodriguez, N. Gaud, V. Hilaire, S. Galland, and A. Koukam. An analysis and design concept for self-organization in holonic multi-agent systems. In S. Bruckner, S. Hassas, M. Jelasity, and D. Yamins, editors, *Engineering Self-Organising Systems*, volume 4335 of *LNAI*, pages 15–27. Springer-Verlag, 2007.

[24] C. Rolland and N. Prakash. A proposal for context-specific method engineering. In *Proc. of the IFIP TC8, Working conference on method engineering on Method engineering : principles of method construction and tool support*, pages 191–208. Chapman & Hall, Ltd., 1996.

[25] 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.

[26] H. A. Simon. *The Science of Artificial*. MIT Press, Cambridge, Massachusetts, 3rd edition, 1996.

[27] I. Sommerville. *Software Engineering*. International Computer Science Series. Addison Wesley, Pearson Education, seventh edition edition, 2004.

[28] SPEM. *Software Process Engineering Metamodel Specification, v2.0, Final Adopted Specification, ptc/07-03-03*. Object Management Group (OMG), March 2007.

[29] W3C. *Web Services Architecture*. World Wide Web Consortium, February 2004.

[30] K. Wilber. *Sex, Ecology, Spirituality*. Shambhala, 1995.

[31] F. Zambonelli, N. Jennings, and M. Wooldridge. Developing multiagent systems: the gaia methodology. *ACM Trans. on Software Engineering and Methodology*, 12(3), 2003.