

Towards an ADL for Designing Agent-Based Systems

Marie-Pierre Gervais

LIP6 and University Paris X

8 rue du Capitaine Scott - F75015 Paris

+ 33 1 44 27 73 65

Marie-Pierre.Gervais@lip6.fr

Florin Muscutariu

LIP6 and University Paris 6

8 rue du Capitaine Scott - F75015 Paris

+ 33 1 44 27 88 61

Florin.Muscutariu@lip6.fr

ABSTRACT

In this paper, we describe the Architecture Description Language (ADL) that we are defining for the design of agent-based systems. This aims at filling the gap between the analysis and design phases in agent-oriented methodologies. The analysis phase enables the description of the software architecture without any consideration of the execution environment while the design phase supplements the analysis output with descriptions related to the distributed environment in which the agents will run. The distributed environment we consider is OMG MASIF compliant mobile agent platforms. Our approach is to use the UML built-in mechanism of extension in order to provide a UML profile for the agent-based systems designers. Thus they can detail the software architecture of the system with information concerning the distribution aspects related to the execution environment.

This work is a part of a project that deals with the development of a methodology for the construction of agent-based systems, called ODAC. This is based on the ISO Open Distributed Processing standards that define an architectural framework for the construction of distributed systems.

Keywords

Design of agent-based systems, Architecture Description Language (ADL), UML

1. INTRODUCTION

A multi-agent system is commonly considered as a distributed system in which the entities are agents. This characteristic requires that these agents be deployed in an environment that supports their execution. Constructing an agent-based system must then encompass all the steps from the analysis and design to the implementation of the system. As the terminology is not standardized, in order to avoid confusion, let us precise what we call analysis and design. The *analysis* relates to the development of a detailed solution that does not depend on achievement choices. It includes the description of

all the processes composing the system operation, the information definition and the tasks' specification. The purpose of the *design* is to lead to the system implementation and to carry out choices to make the models operational for the system achievement. It relates to the operational specifications needed to ensure the achievement of the future system. It includes the taking into account of the means chosen for the adopted solution, namely a target environment that will support the execution of the system.

According to these definitions, it must be recognized that most of the current methodologies focus on the analysis phase in the sense that they define the organization, the architecture and the description of the system elements, namely the agents and their relations [9]. This corresponds to the functional definition of the system, i.e., its software architecture. On the other hand, some works deal with the development of platforms for the agents execution, either in conformance with standards such as MASIF or FIPA or not [10, 4, 8]. However, no correlation is established between the software architecture resulting from the analysis phase and its mapping onto an operational environment. There is a breaking point between the two aspects.

We advocate that the construction of an agent-based system is a process that must take into account the definition of the system as well as its implementation into a target environment in which it will run. This process must be achieved in accordance with an AO methodology that encompasses all the steps including analysis, design and implementation. The methodology must reflect the specificity of the agent technology. The distinction between the agent technology and the other software technologies has been described in [24], based on a unique set of software characteristics. One of the characteristics is a required peer-to-peer protocol. In [25], the distribution of data and control is cited as a reason to consider the multi-agent systems as an important new direction in software engineering. Both the peer-to-peer protocol and the distribution of data and control assume the existence of a distributed infrastructure that provides transparent communication mechanisms (e.g., an agent naming service) or an eventual support for mobility, as identified in [26].

Based on these considerations, LIP6 started a project to develop such a methodology, named ODAC (Open Distributed Applications Construction), using its experience learned from the distributed systems development. For this, it makes use of an ISO standard related to the distributed processing that is the Open Distributed Processing standard (ODP) [11]. This defines a set of rigorous concepts for modeling distributed systems. It makes use of the object paradigm in such a general way that it is possible to deal with ODP objects in the same way as they would be agents. For example, ODP objects are autonomous entities able to act by themselves. In our view, an

agent is an ODP object and agent-based systems are distributed in the ODP term sense, since they comply in a technical and organizational heterogeneous context. They consist of interacting entities, which can be agents and/or objects. The Reference Model of ODP (RM-ODP) can then be useful when building agent-based systems as it provides an architectural framework that encompasses all the needed aspects, i.e. the functional as well as the technical description of the system while providing some correspondence rules between them. However, ODP provides no prescriptive methodology that can be followed in developing a system. Furthermore, the standard supplies no support for this development. For example, it defines concepts and rules, but no notation or formalism is proposed to put in a concrete form the system specification. We then propose in ODAC the use of the UML standardized notation for specifying agent-based systems according to the ODP semantics. Thus, we are defining an Architecture Description Language (ADL) for modeling the system both in analysis and design phases. In a first step of the ODAC methodology development, we have defined the part of the ADL devoted to the analysis phase.

This paper focuses on the design phase of the ODAC methodology. More precisely, it illustrates the requirements for filling the gap between the analysis and the design phases and for achieving this design phase in order to get an operational description of the agent-based system to be implemented. As mentioned previously, the analysis output is the software architecture of the system related to its functional properties. The design activity adds to this description the non-functional properties according to the chosen execution environment. ODAC considers a specific kind of environment, namely a MASIF compliant mobile agent platform. We are then supplementing the ADL already defined in order to include the part devoted to the design phase. Our approach is to define a UML profile enabling the description of the distributed execution environment and the placement of the agents in this environment.

The paper is structured as follows. We first provide an overview of the ODAC methodology. Then we present our approach based on background coming from two fields related to the software design. We describe the UML profile we are defining for the design phase. At the end, we present a conclusion.

2. THE ODAC METHODOLOGY OVERVIEW

A methodology must define a set of concepts, the usage rules of these concepts by organizing them into various steps, the process associated with these steps and a notation. ODAC makes use of RM-ODP concepts. The major one is the “viewpoint” concept, that is a structuring concept of the modeling activity and thereby of the agent-based system modeling. RM-ODP defines five viewpoints: Enterprise, Information, Computational, Engineering and Technology. Each of them focuses on a specific concern when modeling a system and enables to specify the system according to a set of relevant concepts for the preoccupations related to this viewpoint. Therefore this concept of “viewpoint” results from the separation of concerns approach in modeling activity. The complete system specification is then the set of each viewpoint specification with their correspondence rules.

In ODAC methodology, we make use of this concept by associating in an informal way the Enterprise, the Information and Computational viewpoints to the analysis and the Engineering viewpoint to the design [7]. The Technology viewpoint is out of our consideration as it refers to the implementation. ODAC then distinguishes in these two stages intended to define the system those that describe it independently of any target environment of those that describe it according to the environment in which it will be carried out. Two kinds of specifications are identified: the behavioral specification and the operational specification.

The *behavioral specification* results from the specifications established in the Enterprise, Information and Computational viewpoints. It describes the system according to its objective, its place in the company in which it is developed, information that it handles and the tasks that it carries out. For sake of simplicity, we do not mention the complete steps for writing a behavioral specification. Details can be found in [6]. Here, only the part devoted to the Enterprise specification is described. The basic concepts of the Enterprise viewpoint are ODP Enterprise object, role, community, objective, behavior and action [12]. An *ODP Enterprise object* is a model of an entity, either an entity of the system to be specified or an entity of the system environment. ODP Enterprise objects can be grouped to form a *community*. In that case, they exhibit the *behavior* needed to realize the *objective* of the community. By doing this, they fulfill roles of the community since a *role* identifies a behavior. This is a set of *actions* with constraints on when they appear. Actions can be *interactions* between several objects or internal actions. ODAC prescribes the steps for elaborating an Enterprise specification as follows:

- 1) Defining the objective of the system;
- 2) Enumerating all the roles enabling to perform this objective;
- 3) Among these roles, identifying roles of the system environment and roles of the system. This means identifying the S-Community;
- 4) Among the roles of the system, identifying the possible interface roles, i.e. other communities. Assign then to these communities the roles that must be attached to them;

For each community:

- 5) Identifying the Enterprise objects fulfilling the roles of the community;
- 6) Describing the behavior of the community;
- 7) Describing the policies.

To express these various descriptions, we provide the modeler with an ADL based on the UML standard notation. We then have mapped the RM-ODP Enterprise concepts onto the UML ones as illustrated in Table 1 [2].

Table 1. RM-ODP Enterprise and UML Concepts Mapping

RM-ODP Enterprise Concepts	UML Notation
Objective	Use Case
Role	Stereotype of Class “role”
S-Community	Use Case Diagram
Enterprise Object	UML Object
Community	Collaboration Diagram
Community Behavior	Sequence Diagram
Policy	Note

The *operational specification* results from the design step corresponding to the projection of the behavioral specification on a target environment reflecting the real execution environment. It depends on the specification established in the Engineering viewpoint, which describes the execution environment. It constitutes the description from which code is generated and the implementation is carried out. We are then supplementing our ADL used in the analysis step in order to include the design concerns. For this, we make use of background coming from two areas related to the software design.

3. RELATED WORKS

We aim to fill the gap existing between the analysis phase and an agent execution environment. We want to provide the designer with a notation for describing the operational specification that is the mapping of the software architecture resulting from the analysis phase onto an operational environment.

The notation for the design step must include concepts related to the distribution aspects while enabling the description of the considered environment. The RM-ODP Engineering viewpoint provides such concepts as it focuses on required mechanisms and functions supporting distributed interactions among RM-ODP objects in the system. The definition of an ODP object as a model of an entity permits us to consider an ODP object as an agent. The agent execution environment we consider is in conformance with the MASIF standard. We have then mapped the MASIF standard concepts onto the RM-ODP Engineering concepts [15]. We refer them from now as the MASIF *platform elements* (e.g. region, agent system, place, agent, etc) [21]. In order to ensure continuity between analysis and design steps, the ADL we are defining for the design step must also be based on the UML notation.

We have investigated two fields that address issues relevant for our objective, namely the definition of an ADL for the description of an ODAC operational specification:

1. The *Object Engineering* field addresses the distribution aspects description through the use of UML. Moreover, some work has been done in considering object engineering on an RM-ODP base;
2. The *Architecture Engineering* field provides concepts to characterize the elements to be designed together with their relations. Since agent semantics is richer than the object semantics, Architecture Engineering represents a

valuable work enabling the UML representation of different element semantics.

We detail hereafter works related to these two fields.

3.1 Object Engineering

Object engineering has made some important progress since the appearance of UML. UML seems to be generally accepted as the language to describe object architectures. It is the result of a unified effort of other object languages (OMT, OOSE and OODA). Basically, the notions that UML introduces can be grouped in three categories:

- Things;
- Relationships;
- Diagrams.

Although most of the efforts are devoted to the analysis phase of an object-oriented system, there are also possibilities to describe distribution issues (i.e. node, component, deployment diagram, etc). But UML lacks a methodology that defines the links between various elements.

Probably the most known methodology, which implicitly establishes a part of these links is the RUP (Rational Unified Process) developed by Rational Corporation [19]. This permits to describe some distribution concerns and is based on UML concepts as defined in UML Meta-Model.

Other projects consider their own object-oriented methodologies based on an RM-ODP approach.

a) The TRUMPET project consisted in modeling an RM-ODP system. This was done using a UML notation [16]. Most of the project was concerned with the analysis phase, although an extension of UML is presented for the RM-ODP engineering concepts dedicated to a chosen environment. This extension uses the UML subsystem considerations presented in [20]. In addition, the project is only concerned with UML deployment diagrams. Moreover, an RM-ODP object is considered to be an object in the usual meaning in object-oriented modeling, although its definition enables to map richer semantics on it.

b) Other efforts use an RM-ODP approach for UML object-oriented modeling but are only interested in the analysis phase [17]. More complete approaches never really passed the draft stage [18].

c) Lately, the DOT Profile project has proposed to consider an RM-ODP system on a component basis but for now it only addresses the analysis phase [3].

3.2 Architecture Engineering

Architecture Engineering mainly consists of ADLs definitions enabling the description of architectures. Basically, ADLs have two main concepts: components and connectors. An architecture is then described as components connected by the connectors. The graphical representation for ADL is boxes for the components and lines for the connectors with specific semantics and properties. Few means are offered to describe the so-called non-functional properties of the components and connectors, i.e. properties that take into account an execution environment [13]. Most of the effort is dedicated to the analysis phase. Two main approaches are used in ADL-based methodologies for mapping the software architecture described in the analysis step onto the execution environment (Figure 1):

- a) a *compilation* approach when the architecture is mapped directly onto a specific executing environment. An example of this approach could be seen in the OLAN project [1]. The compilation is done using an “OLAN configuration machine” which makes all the implementation choices.
- b) the *construction* of a framework over the executing environment capable to match the ADL’s semantics (**bottom-up construction**). An example of this approach could be seen in C2 [14]. C2 constructs a class hierarchy that implements the component and the connector classes.

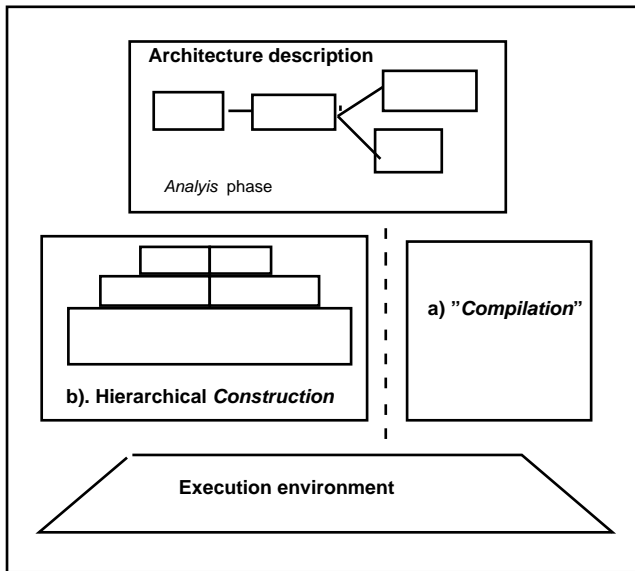


Figure 1. Two approaches for implementing the architecture description.

Methodologies used for the architecture description based on particular ADLs representations have been supplemented lately with a UML approach. Instead of using specific ADL’s boxes and lines, UML extensions are defined for capturing the ADL’s semantics [5]. An example is the C2 ADL mapping to UML [14].

This trend of UML extension and adaptation for different purposes is encouraged by OMG and some *Profiles* are already made available [22].

3.3 Summary

We are defining a UML-based ADL for the *design* phase of a agent architecture system. RM-ODP offers the overall framework for that. The **object-oriented modeling** helps us to consider the *platform elements* representation using the existing UML concerns for distribution.

Since the *agent platform elements* have other semantics than the UML standard can offer, we have to choose:

- to create a different Meta-Model by operating modification at this level of abstraction,
- or to use the built-in mechanism of extension in order to obtain the required semantics.

Since the scope of UML is a “Unified Language”, rather than modifying the Meta-Model, we propose to use the UML extension mechanism by introducing tagged values and

semantics constraints expressed in OCL and grouped them in a UML profile.

The Architecture Engineering offers a feedback for different semantics representations with UML. Also architecture engineering permits us to situate our approach for the design phase as a **top-down construction**. The construction offers the possibility to refine the behavioral specification with the imposed choices and characteristics related to the distributed execution environment.

4. UML MASIF-DESIGN PROFILE

The ADL that we propose for the design phase is defined as a UML profile. This is called “MASIF-DESIGN” profile, as for now, the distributed execution environment we consider is in conformance with the OMG-MASIF standard.

MASIF presents a minimum set of *concepts* and *operation* interfaces necessary for interoperability. The term *operation* in this context has a UML meaning. These concepts and operations are considered in an overall RM-ODP framework. The *operation* is the *function* equivalent in an ODP context. RM-ODP guide us in establishing the operations required to manage physical distribution, communication, processing and storage. Some of the operations defined in MASIF accomplish requirements for the management of this physical distribution. MASIF presents these operations grouped in two IDL interfaces: MAFFinder and MAFAgentSystem. These interfaces present the prototype of the methods that implement these operations.

The ADL for the design phase must enable the description of the considered environment. Thus we have first to model the MASIF platform elements with UML diagrams. As this model permits us further to locate agents in a distributed environment, two issues must be considered:

- The placement of agents in the distributed infrastructure;
- The representation of distribution transparencies.

A distribution transparency is the capacity of hiding the distribution aspects in the behavioral specification elaborated in the analysis phase. The platform elements cooperate to provide a transparency by bringing uniformity to some aspects of agents’ distribution (e.g. uniformity of naming whatever the location of the agent). RM-ODP presents an inventory of assumed transparencies for a distributed system (e.g. access transparency or location transparency). MASIF presents a limited set of operations available to model the interactions between the platform elements needed for the transparency specification.

The transparencies have to be specified as analysis phase requirements. They enable to refine the existing behavioral specification with introducing additional behavior, including the use of one or more operations of the platform elements.

4.1 The placement of agents in the distributed infrastructure

The placement of agents in a distributed infrastructure is specified with UML deployment diagram. In order to do that, we have to model the infrastructure platform elements with UML diagrams. These platform elements are organized in a hierarchical configuration. First there is the *Region*, which

makes the link among *Agent Systems*. An *Agent System* includes *Places* where *Agents* reside and a *CoreAgency* as a management module.

This modeling process is not always straightforward. Each platform element can be represented at the *type level* and the *instance level*. An *instance* of a *type* is defined in ODP as a $\langle X \rangle$ (anything) that satisfies the *type*. A *type* is a predicate that characterizes collections of $\langle X \rangle$ s. The same approach of type and instance is referred in [5]. For simplicity, an analogy with object modeling concepts would be classes for *types* and instances of these classes.

We model the platform elements in the MASIF-DESIGN profile with stereotypes illustrated in Table 2.

Table 2. The stereotypes modeling the MASIF platform elements

MASIF platform elements	Type level UML Meta-model Class	Instance level	Stereotype Name
Region	Stereotyped Subsystem	Stereotyped Subsystem instance	Region
Agent System	Stereotyped Node	Stereotyped Node instance	Agent System
Core Agency	Stereotyped Subsystem	Stereotyped Subsystem instance	CoreAgency
Place	Stereotyped Package	Stereotyped Component instance	Place
Agent	Stereotyped Component	Stereotyped Component instance	Agent

a) The *Region* fully interconnects agent systems and enables the point-to-point transfer of information between them. The Region has a region registry with all the information about the places and agents. We represent Regions as stereotyped subsystems (Figure 2).

In UML, the subsystem is both package and classifier. A package is a grouping of model elements. A classifier is a *super-type* (abstraction) used to specify the common characteristics of the UML *Class*, *Interface* and *DataTypes*. A classifier describes behavioral and structural characteristics. The subsystem is instantiable.

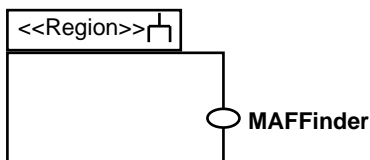


Figure 2. The stereotyped subsystem “Region”

A subsystem groups three elements: *operations*, *specifications* elements and *realization* elements. The services provided by the subsystem are represented by interfaces and the corresponding operations. The specifications and realization elements detail the internal behavior in order to assure these services and the elements that realize this behavior.

In MASIF, the region has an IDL interface named the MAFFinder. This is a set of operations needed to implement a “white pages” service. Thus graphical representation of the Stereotyped subsystem “Region” can be refined as illustrated in Figure 3. Only a few operations are listed here, the complete list can be found in the MASIF specification. It should be noted that the specification elements and realization elements of a Region are not addressed in MASIF. Thus each mobile agent platform that implements the MASIF standard has to make the desired choices.

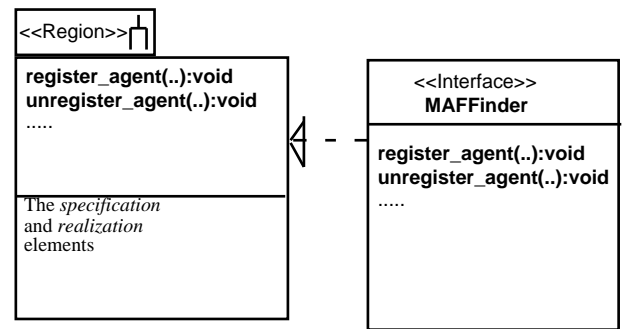


Figure 3. Refinement of the stereotyped subsystem “Region”

b) The *AgentSystem* is the platform that can create, interpret, execute, transfer and terminate agents. We represent it as a stereotyped node. A node has an instance. The graphical representation is illustrated in Figure 4.

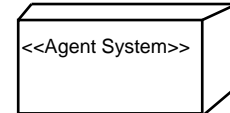


Figure 4. The stereotyped Node “Agent System”

Each agent system has one *CoreAgency*, represented as a stereotyped subsystem.

c) The *CoreAgency* supports the management services provided to agents in an Agent System. The CoreAgency has an interface defined in MASIF as MAFAgentSystem. Only a limited number of operations are described. The complete list of operations can be found in the MASIF specification. The specification elements and realization elements of the Core Agency Subsystem stereotype are not addressed in MASIF. The graphical representation and the implementation diagram are the same as the *Region* ones.

d) A *Place* is a context within an *Agent System* in which an agent can run. It can provide functions such as access control. We represent it as a stereotyped package (Figure 5).



Figure 5. The stereotyped package “Place”

Since a package has no instance, we represent the place instance as a component instance. The component *realizes* the place. Figure 6 illustrates the corresponding component diagram.

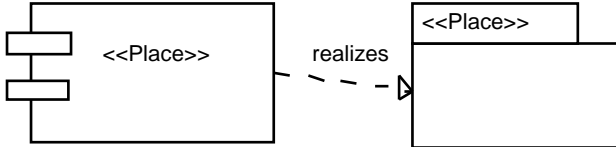


Figure 6. The Place component diagram

e) We represent an *Agent* as a stereotyped component. The component *realizes* the stereotyped class that represents an agent in the analysis phase [23]. Additional information needed for the implementation can be included in the component diagram related to an agent implementation. For example, Grasshopper, which is a MASIF implementation platform, introduces an interface definition that specifies the agents' interaction points, i.e. the public operations (Figure 7).

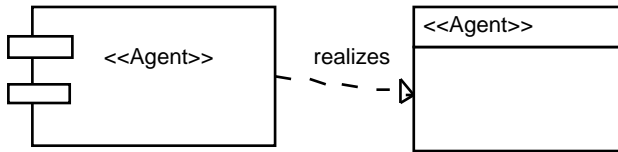


Figure 7. The Agent component diagram

A component instance is defined in UML as a run-time implementation unit and may be used to show implementation units that have identity at run-time, including their location on nodes. The vast semantic of this definition permits us to consider the instance of a stereotyped package “Place” and of a stereotyped component “Agent” as components instances but with different semantics. The Place is a grouping unit of different agents. As for the Agents, they are the basic executable units. Components are things that participate in the execution of a system and nodes are things that execute components.

The dependency relations among the platform elements are represented in a component diagram (Figure 8).

The deployment diagram represents the physical distribution of the platform elements in a hierarchical structure. The diagram is static and represents an image of the system (Figure 9). We use transition stereotypes already defined in UML 1.3. These stereotypes permit us to specify all the locations that an agent visits during its lifetime and the eventual clones that it creates (Table 3).

As this representation could lead to an overloaded diagram, then we choose to multiply the deployment diagrams, one for small group of agents when it is necessary. This type of

deployment diagram helps us to consider further some of the distribution transparencies.

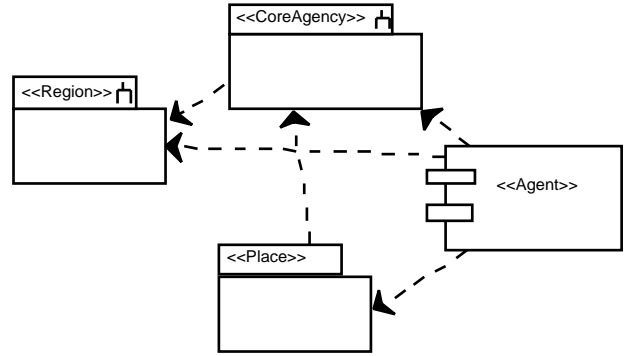


Figure 8. The platform elements dependencies

Table 3. Transition stereotypes

UML Stereotype	Description
Become	Migration of an agent
Copy	Replication of an agent

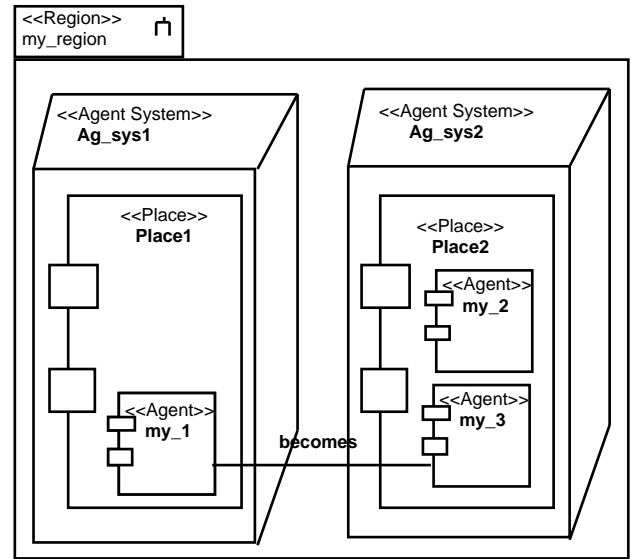


Figure 9. The placement of agents in the hierarchical structure

4.2 Transparencies

The transparencies defined in RM-ODP are: access transparency, failure transparency, location transparency, migration transparency, relocation transparency, replication transparency, persistence transparency and transaction transparency. Our ongoing work only deals with location and access transparencies. The others are for further study.

a) Location transparency masks the use of information about location in space when identifying and binding interfaces of agents. Thus, agents can interact with other agents without using the location information.

There are two location issues, namely the location of an agent and the location of resources files for an agent.

In MASIF, the *location* is defined as the path to an agent system based on the agent system, the agent or the place. The operation named *MAFFinder.lookup_agent()* permits to connect two agents without knowing their location by requesting the region about the agents' location. The *location* descriptor in MASIF is either a URI containing a CORBA name or a URL containing an internet address.

We have also to consider the location of the agent definition file. An Agent System that executes an instance of an agent needs this agent definition. The operation *MAFFAgentSystem.fetch_class()* permits to retrieve this definition from the location of this file provided by the designer. In order that the designer can locate the source file with the agent definition, we introduce a tagged value named *FileDefinitionLocation*.

b) Access transparency masks differences in data representation and invocation mechanisms to enable interactions between agents. Here, there are also two issues, namely the link establishment of the communication and the security aspects in terms of access rights.

For the link establishment, MASIF refers existing mechanisms such as RMI, CORBA or RPC. However nothing is established in order that the designer makes the choice between these mechanisms. Some implementations, such as Grasshopper detail further the possibility to make these choices. We provide a tagged value *Link* so that the designer identifies the link he chooses.

As for the access rights, MASIF defines the notion of Agent's *authority*. An agent's *authority* identifies the person or organization for whom the agent acts. An authority must be authenticated. MASIF discusses the requirements for a secure mobile communication. The only operation related to access rights is *MAFFinder.list_all_agents_of_authority()*. In order that the designer identifies the authority of an agent, we introduce a tagged value named *Authority*.

5. CONCLUSION AND FUTURE WORK

In order that designers of agent-based systems describe the model resulting from the analysis phase according to an execution environment, we provide them with an ADL. This enables the designers to describe the mobile agents platform they use and to locate the agents identified in the analysis step in this description. This ADL is defined as a UML profile since we make use of the built-in mechanism of extension. Thus it is based on UML notation.

Current work has identified the stereotypes needed to describe an OMG MASIF compliant mobile agent platform and its hierarchical structure and to locate the agents in such a structure. It also provides means in terms of tagged values enabling the designers to represent location and access transparencies. Further work is devoted to the refinement of our proposal by identifying attributes to characterize the platform elements and the transparencies not addressed in this paper. In addition, the approach described here in the definition of an ADL for the design of agent-based systems will be applied to another application domain, namely the active networks.

REFERENCES

- [1] Bellisard L. et al., *Distributed Application Configuration*, In proceedings of the 16th International Conference on Distributed Computing Systems, IEEE Computer Society, Hong Kong, May 1996, pp579-585
- [2] Blanc X., Gervais M.-P. and Le Delliou R., *Using the UML language to express the ODP Enterprise Concepts*, In proceedings of the 3th International Enterprise Distributed Object Computing Conference (EDOC'99), IEEE Press, Mannheim, Germany, September 1999, pp50-59
- [3] Born M., Holz M. and Kath M., *A Method for the Design and Development of Distributed Applications using UML*, International Conference on Technology of Object-Oriented Languages and Systems (TOOLS Pacific), Sydney Australia, November, 2000.
- [4] FIPA-OS, <http://fipa-os.sourceforge.net>
- [5] Garlan D. and Kompanek A.J., *Reconciling the Needs of Architectural Description with Object-Modeling Notations*, UML'2000.
- [6] Gervais M.P., *ODAC : une méthodologie de construction de systèmes à base d'agents fondée sur ODP*, rapport LIP6 2000/28, November 2000 (in French)
- [7] Gervais M.P., *ODAC: An Agent-Oriented Methodology Based on ODP*, Submitted in Journal of Autonomous Agents and Multi-Agent Systems.
- [8] Glass G., *Overview of Voyager : ObjectSpace's Product Family for State of the Art Distributed Computing*, CTO Object Sapce, www.objectspace.com, 1999
- [9] Iglesias C. A., Garijo M. and Gonzalez J. C., *A survey of agent-oriented methodologies*, In Proceedings of the 5th International Workshop on Agent Theories, Architectures and Languages (ATAL'98), LNAI n°1555 - Springer Verlag, Paris, France, July 1998, pp317-330
- [10] IKV++ GmbH, *Grasshopper, A platform for mobile software agents*, www.ikv.de/products/grasshopper
- [11] ISO/IEC IS 10746-x — ITU-T Rec. X90x, *ODP Reference Model Part x*, 1995
- [12] ISO/IEC CD 15414, *ODP Reference Model : Enterprise Viewpoint*, January 2000
- [13] Medvidovic N. and Taylor R.N., *A Classification and Comparison Framework for Software Architecture Description Languages*, IEEE Transactions on Software Engineering, 26(1), January 2000
- [14] Medvidovic N., Egyed A., Rosenblum D.S., *Round Trip Software Engineering Using UML: From Architecture to Design and Back*, Proceedings of the Second International Workshop on Object-Oriented Reengineering (WOOR'99), Toulouse, France, September 6, 1999.
- [15] Muscutariu F. and Gervais M.-P., *Modeling an OMG-MASIF Compliant Mobile Agent Platform with the RM-ODP Engineering Language*, in Proceedings of the 2nd International Workshop on Mobile Agents for Telecommunication Applications (MATA'00), Lecture Notes in Computer Science n°1931, Springer Verlag (Ed), Paris, France, September 2000, pp133-141

- [16] Kandé M.M, Mazaher S., Prnjat O., Sacks L., Witting M., *Applying UML to Design an Inter-Domain Service Management Application*, OOPSLA'2000.
- [17] Milosevic Z., *ODP viewpoint languages and UML: a case of study*, <ftp.dstc.edu.au/AU/staff/zoran-milosevic.html>
- [18] Miller J., *Relationships of the Unified Modeling Language to the Reference Model of Open Distributed Computing* version 1.1.2, 21 September 1997.
- [19] *RationalRose* (www.rational.com)
- [20] Miller J. and Wirfs-Brock R., *How can subsystems be both a package and a Classifier*, (www.omg.org)
- [21] OMG *MASIF Standard* (orbos/98-03-09) <http://www.omg.org>
- [22] *OMG Unified Modeling Language Specification*, Version 1.3, Mars 2000, <http://www.omg.org>
- [23] Odell J., Van Dyke Parunak J. and Bauer B., *Extending UML for Agents*, AOIS Workshop at AAAI 2000.
- [24] Petrie C., *Agent-Based Software Engineering*, AOSE 2000.
- [25] Wooldrige, M., and Ciancarini P. *Agent-Oriented Software Engineering: The State of the Art*, AOSE 2000.
- [26] Shehory, *On. Software Architecture of Multi-Agent Systems*, AOSE 2000