Agent Metamodel for virtual reality applications

Ronan Querrec, Cédric Buche, Frédéric Le Corre, Fabrice Harrouet

Agent Metamodel for virtual reality applications

Ronan Querrec, Cédric Buche, Frédéric Lecorre, and Fabrice Harrouet

UEB/ENIB/CERV,
25 rue Claude Chappe
F-29490 Plouzané France {Querrec,Buche,Lecorre,Harrouet}@enib.fr

Abstract. The various existing agent models do not cover all the possible uses we consider for virtual reality applications. In this paper, we present an agent metamodel (BEHAVE) based on an environment metamodel (VEHA). This metamodel allows defining agents and organizing teams of agents in a virtual environment. The use of this metamodel is illustrated by the GASPAR application which simulates activities on an aircraft carrier.

Keywords: agent, metamodel, virtual reality

1 Introduction

In the context of virtual reality, many applications are based on multi-agent systems to simulate human activities or to simulate the environment reactions to users actions. These applications use various agent models, multi-agent system and platform as JADE [14], JACK[1] or GAIA [16]. Several studies attempted to generalize these models and propose agent or multi-agent system metamodels [6, 2]. Multi-agent systems are used to simulate human activities, physical or biological systems; thus, it appears difficult to propose a metamodel to cover all of these uses while keeping an effective language for the designer. Moreover, thoses agents metamodels focuses on the agent model but not it's environment. However in the case of virtual reality application the definition of the environment is an important task that must interact with agents modelisation. We distinguish two majors uses of multi-agent systems. First multi-agent systems to simulate physical or biological phenomenon like in [3] and second multi-agent systems to simulate human activities. In this article we focuses on the latter use.

Such applications still keep difficult and time consuming to develop. Classical uses of those type of applications are simulations, communication, training and teaching which exhibits functionalities that can be developed independently from the specific domain they apply to. Our goal is to provide an highter level of abstraction in the conception of virtual reality applications. The conception model of a specific application become then datas for our generic virtual reality metamodel (in the context human activities simulation as seen before). As a consequence we provide a langage wich permits to an domain expert to define

[1] http://agent-software.com

the environment he adress as well as the activities that are executed in this environment. This description permits to automatically create the simulation in a virtual reality application and is seen as a knowledge base to agents executing the activities in the environment.

We propose MASCARET, a metamodel to describe virtual environments and the agents evolving in this environment. This metamodel provide an unified modeling langage to describe the structure of the environment (entities, positions...), entitie's and agent's behavior. MASCARET is founded upon UML[2]. UML has already been used by agent's metamodels to describe agent's activities like [1], but the major contribution of MASCARET is the strong link between environment design and agents activities design.

In this article we focuses on the agent metamodel, but first (section 2), we describe the principles, the workflow and the bases to create a MASCARET application. In section 3, we present our proposition of agent's metamodel for human activities simulation in a virtual environment. As an example of MASCARET use, the GASPAR application which simulates activities on an aircraft carrier is presented section 4.

2 The Mascaret metamodel

The aim of MASCARET is to provide a metamodel to describe the virtual environment (VE), not in terms of geometric space, but by providing the semantics required for the artificial agents or humans to be able to construct for themselves a representation of the environment and to act together to reach their goals. MASCARET metamodel is based on UML and enables the construction of domain patterns from virtual environments and from the corresponding *concrete* virtual environments. However, UML metamodel does not allow us to define the specific concepts of virtual reality. In MASCARET, we propose to extend UML in order to represent these concepts.

Agents need to know which objects make up the virtual environment, how to access them, their properties, their behavior and how to interact with them. Three kinds of knowledge can be expressed using MASCARET:

– Domain concepts. This entails the semantic description of the concepts relating to the field of activity concerned. Knowledge of the domain is expressed both at the model (concept) level (call M1), and at the level of the occurrences of these concepts, call M0 (tangible objects populating the environment). In MASCARET as in UML, this knowledge is represented by classes and instances. This provide the onthology of the EV in 2 abstraction levels like OWL[13].
– The possibility of structuring and interacting with the environment. These concepts looks like those suggested in *smart objects* [9] which reify those properties required for interactions. The means available to the user or to agents must be specified in order to modify the environment. Within the

[2] http://www.omg.org

context of virtual environments, most of the tangible objects within those environments have a geometric representation, and are situated within the environment. These objects are entities and all have the properties of the instances classes as well as geometric, topological properties and animations.
– Entities' behavior. Within the framework of a VE, the environment's reactions to the user's actions must be simulated. The MASCARET entities have reactive behaviors which are triggered by events that can be caused either by the user, by agents or by another of the virtual environment's entities. Those behaviors are defined by UML *StateMachines*. Entities' behavior and its execution represents also an element of agent's knowledge.

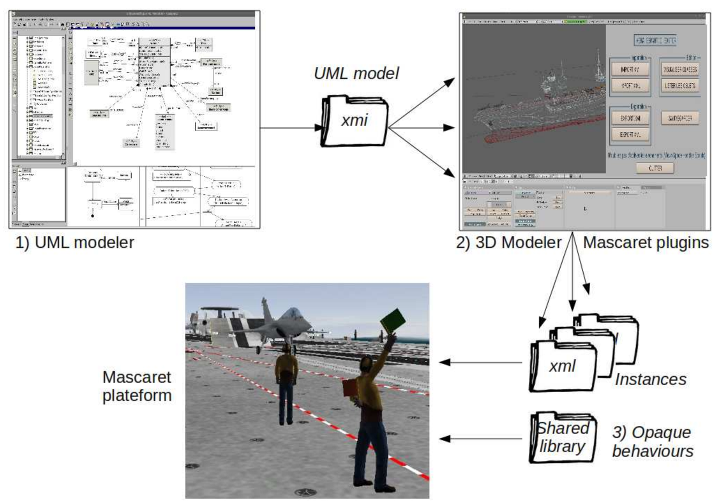

Fig. 1. Process to develop applications using MASCARET

All applications designed with MASCARET follows the process illustrated in Figure 1. First, the domain expert defines the virtual environment's model (M1 model) in the form of UML–MASCARET diagrams exported into XMI. He has to describe the class models and behavioral models (state machines and activities) and the human activities using UML collaboration and activities diagrams. This step is completed using a UML modeler which supports metamodels defined as UML profiles.

Second, 3D designers have to construct geometrical objects (in *Vrml* format). This mean the construction of shapes and definition of geometries (informed points, interaction surfaces and volumes) using classical 3D modeler. A MASCARET plugin is added to 3D modeler in order to refer the UML model (XMI file) and then add semantics to geometrical objects which are then defined as instances of the domain model (M0 model) Many virtual environments can thus be constructed based on the same M1 model.

Third, computer scientist has to code the possible *opaque behaviors* for specific non-introspectable behaviors. At the end, the user has to launch the simulation platform : loading (M1) domain models and specific environments (M0), and activating the interaction and immersion devices.

3 The agent metamodel

In the previous section, we justified our choice to use UML as a language to describe the concepts of the specific fields and their relationships. These same comments are applied for our metamodel of multi-agent system. In addition, we use multi-agent systems to simulate human activities that are highly contextualized by the environment. It is therefore necessary to use the same language to describe activities and to describe the environment, as these operations manipulate the virtual environment.

Several agent models or agent metamodels were proposed in the literature using UML. These models can be based on the UML metamodel [1] or propose a behavior agent model automatically interpreting knowledge expressed in UML [8, 15, 4]. However, the proposed models cover only parts of the kinds of behavior that interest us, and especially do not incorporate environment modeling which supports these behaviors.

Moreover, FIPA[3] offers models that claim to be a standard for agent modeling. Our model is inspired by this standard. Therefore, we do not propose a completely innovative agent model, but an operationalization of concepts existing in other models for virtual reality applications. Concepts involved are: the agent, its actions or behaviors (section 3.1), its means of communication (section 3.2) and the organizations (section 3.3). Figure 2 present an overview of the proposed agent metamodel.

3.1 Agent and behavior

The agent model we propose is inspired by the FIPA standard and its implementation in JADE. We implement the proposed concepts by extending MASCARET. An agent performs behaviors and can communicate with other agents through messages. Agents are hosted by a platform, it is not possible to obtain a reference (in the computer sense) to an agent. An agent is identified by its name and the platform on which it is hosted.

An agent is an instance and has a type in the same manner as entity and class. The `AgentClass` class describes the different types of agent fields. Thus, it is possible to describe the properties, statements and actions of agents.

This model suffices to address the different generic kinds of fields. This also means that the model we propose is not intended to require inheritance from

[3] http://www.fipa.org

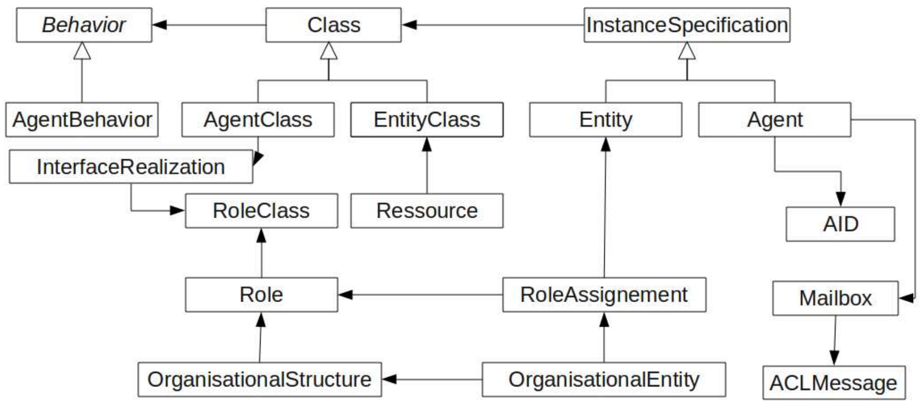

Fig. 2. Overview of the agent metamodel.

`Agent`. The specificities that should be obtained by deriving `Agent` are actually formulated in our model by properties, operations and specific behaviors (new instances of `AgentClass`).

To describe the agents behavior we implement the model proposed in JADE. The behavior calls the `action()` method while a condition is not met. To help designing a behavior, JADE provides `OneShotBehavior` which is executed once and `CyclicBehavior` looping forever. The agent then conducts a set of activities which are arranged in sequence. The execution behavior (calling the `action()` method) is managed by the scheduler proposed by MASCARET. The user then provides new behaviors by deriving `OneShotBehavior` or `CyclicBehavior` in order to overload the `action()` method. The execution behavior is managed by the scheduler proposed by MASCARET and this execution is also an explicit knowledge (start, and result...) that can be used by agents.

3.2 Communication

he agents use messages to communicate with each other. We then use the model proposed by FIPA:ACL (Agent Communication Language[4]). A message is represented by a performative. The ACL model proposes 23 performatives. For example, an agent uses the REQUEST performative to make a request to another in order to obtain the value of a property or to make it execute an action. The INFORM performative allows an agent to inform another of the value of a property or to confirm the execution of an action in response to a REQUEST. The messages are expressed in a language and cover an onthology. Several languages exist for this purpose but we use the one proposed by FIPA : FIPA-SL. Each agent has an automatic communication behavior. This communication behavior is a `CyclicBehavior` which reacts to every new incoming message. The purpose

[4] Specification FIPA SC00061

of this behavior is to automatically analyze the message content according to the performatives. For now, we only consider REQUEST and INFORM. In the language FIPA-SL we manage everything that relates to the achievement of an action. Thus, it is possible for an agent to request the execution of an action to another agent. For example, the following message is received by agent1 asking him to perform the action `openDoor`.

ACLMessage : ((action (agent1 (openDoor (door_right)))))

By default the communication behavior introspects the content of the class of the receiver. If the requested action name is found, the agent executes this operation. The execution of the operation may require parameters. They are assigned to the parameters passed in the ACL message depending on their type. If no operation is found, then the behavior looks for a procedure with that name in the organizations in which the agent plays a role. If it exists, then the agent triggers the execution of this procedure, using the necessary resources for this procedure as parameters. If an action or a procedure shall be conducted on the occurrence of this message, then the agent responds a AGREE performative to the sender of the message. If no action and no procedure is found or is achievable (depending on the state of the environment) then the agent responds with a NOT UNDERSTOOD performative. This way of responding is normalized by the FIPA standard. Request and informations on the value of properties are also possibles. All those functions are possible in the context of an automatic and generic agent behavior thanks to the fact that all informations are explicit in the simulation.

3.3 Organisation

Because they deal with the simulation of human activity by collaborative multi-agent systems, the notions of collaboration or organizational structure interactions between participants are important. The organization can be an *a priori* description or an *a posteriori* inference. It can be defined according to strict rules of behavior or coming from agents. In the context of the simulation of human activities in a virtual environment, the modeler explicitly describes the structure of the organization.

Several organizational models exist [10, 12, 11, 5], but in each of them the concept of group, organization or collaboration takes place in the concept of role. In general, the organization aims at structuring the roles. A role may include the description of the responsibility of the agent playing the role or a list of actions performed by the agent. In [7] the role describes the rights and duties of an agent.

As for the environment or for the agents, the organization can also be described in terms of its structure and in terms of instances of this structure. The organizational structure describes the roles that composes the organizational entity as described when assigning roles to agents.

Finally, the description of these organizations by the modeler can not be independant upon the environment and agents. Since they are described in UML, it appears necessary to describe these organizations in the same language. Our approach is then to interpret the UML collaboration diagrams to instantiate the elements of the organizational model we propose.

An organizational structure (`OrganisationalStructure`) describes how concrete organizations are instantiated. This is the same approach as the principle of COLLABORATION in UML. In MASCARET, a role is a set of action. The principle is to give constraints to ensure that the agent plays a role using skills. We represent this principle with the concept of `RoleClass`. A `RoleClass` is a kind of `Interface` (with the same meaning as in UML). This allows us to describe all the actions an agent has to know without describing how they are actually executed. As seen before, an `AgentClass` describes the agent structure, its statements and its possible actions. It uses also an `InterfaceRealization` to implements a `RoleClass` (inheriting from `Interface`). This is substantially the same principle as in UML. This helps provide a rich mechanism on how a service interface is realized. For example an action of the interface can be achieved in an `AgentClass` by a complex arrangement of actions. Organizations and roles may have the responsibility for resources (`Ressource`). This allows linking with the environment in which the organization operates. The concept of resource can be described independently of the concrete object. A resource is defined by its name and the entity class that can play the role of this resource.

An organizational entity (`OrganisationEntity`) is an instance of an organizational structure. This is the same approach as the principle of COLLABORATIONUSE in UML. It assigns roles to agents (`RoleAssigment`) and resources to entities (`ResourceAssignement`). There are several organizational units for the same organizational structure. Roles and resources can be set *a priori* but could also be dynamic. In BEHAVE, a XML file type can instantiate the entities of the environment, the agents and also the organizational entities. It is then able to create an organizational entity, specifying the structure and its corresponding complete assignment of roles and resources, to agents and entities. Of course this can be done dynamically during simulation and also be changed. It could also have been possible to define a `CollaborationUse` in the UML modeler to perform this instantiation; however, it seems reasonable to separate the model description and its environment. Indeed the same model can be used to describe several environments and an UML modeler is not the ideal tool to place objects or agents in a 3D space.

4 Virtual reality application

GASPAR is a virtual reality application developed to simulate human activities on an aircraft carrier. In GASPAR, a typical scene, such as the one shown in figure 3, is made up of around 1,000 entities, each with a 3D representation (VRML), i.e. a total of 1 million polygons. In this scene, there are around 50 agents, divided into 10 teams, each with an average of 5 roles. Each of these teams is responsible for an average of 5 procedures. The most complex procedure activates 9 roles and organizes 45 actions. In this scene, at each moment, around 50 behaviors are activated. It is implemented using AReVi[5] and runs in real-time (around 40 frames per second) on a desktop computer with 2GB of RAM, a 64 bit processor running at 1.3 GHz, and a GeForce GPU with 1GB of video memory.

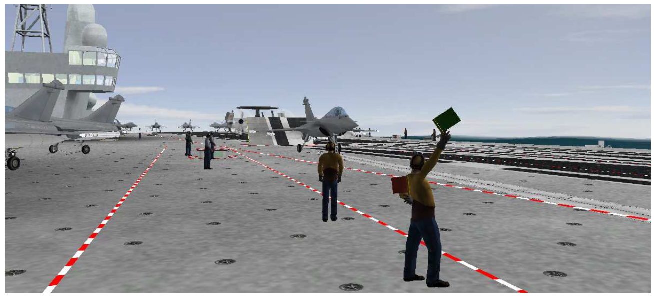

Fig. 3. The Gaspar application.

This application uses the generic models presented in the previous section, *i.e.* the structure of the environment, objects, organizations and procedures present in the application are described by a Uml model. Figure 4 represents the global architecture of the model used in Gaspar.

In this figure, we can see that the model is divided into three packages: the *Organizations* package, the *Environment* package and the *Agent* package.

- The *Environment* package describes all the kinds of objects (classes) that compose the environment. Links between classes are also represented as we can see on figure 4. This package is interpreted by Veha.
- The *Agent* package represents the different types of roles that an agent will take. Those roles correspond to the those which are defined in the real procedures of catapult-launching or landing for example. A role is made up with several methods which represent operations that the agents are able to execute. An agent can be unable to execute some actions that another agent is responsible for (notion of competence). That is why "Staff" class is derived in several subclasses, representing specialities of different types of staff members on the aircraft carrier for example.
- The *Organizations* package describes the different teams on the aircraft carrier, the roles that compose those teams, and the procedures that those teams can execute. Roles that take part in those procedures correspond to the types of agents defined in the *Agent* package. Figure 4 shows the activity diagram representing the lift-off procedure of an helicopter from the aircraft carrier. Two agents are involved in this procedure: the pilot of the helicopter and an agent which is of type PEH, playing the role of ChefPEH.

The French navy (DCNS) provides scenarios, pre-calculed by a scheduling and resources management tool. The Gaspar application makes possible to

[5] `http://svn.cerv.fr/trac/AReVi/`

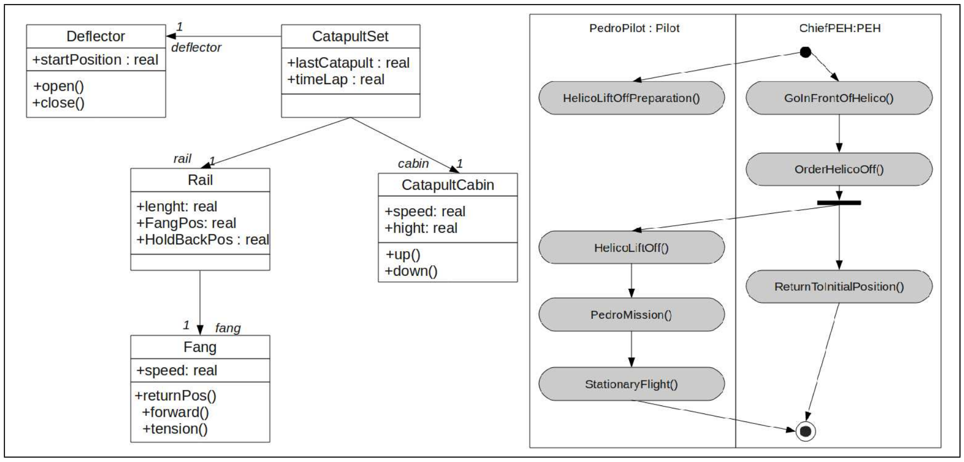

Fig. 4. UML model describing the GASPAR application

replay those scenarios in order to estimate the compatibility of the functional requirements and the geometry of the ship.

5 Conclusion and futurs works

In this paper, we presented an agent metamodel (BEHAVE) based on an environment metamodel (MASCARET). The metamodel allows the integration and the management of complex teams of agents in an interactive virtual environment. We saw the metamodel use in the GASPAR application which simulates activities on an aircraft carrier.

The FIPA standard proposes to provide a knowledge base to the agent, but without giving a formalism for the knowledge base. As a perspective of our work we propose that the agent knowledge base could be a subset of the environment. Thus it would be possible to drive communication to read or write in this knowledge base according to the FIPA-SL messages received. A behavior specifically developed for the application will then only manipulate this knowledge base. Several problems remain, however. How to determine the information the agent has at the beginning of simulation? Could all behaviors really be expressed in these terms? How to synchronize the modified knowledge base and the reaction behavior concerned?

References

1. B. Bauer, J. Muller, and J. Odell. Agent uml: A formalism for specifying multiagent software systems. In *Agent-Oriented Sofware Engineering*, volume 1957 of *Lecture Notes in Computer Science*, pages 109–120. Springer Berlin / Heidelberg, 2001.

2. G. Beydoun, G. Low, B. Henderson-Sellers, H. Mouratidis, J. Gomez-Sanz, J. Pavon, and C. Gonzales-Perez. Faml : A generic metamodel for mas development. *IEEE Transactions on Software Engineering*, 35(6):841–863, 2009.
3. G. Desmeulles, S. Bonneaud, P. Redou, V. Rodin, and J. Tisseau. In virtuo experiments based on the multi-interaction system framework: the réiscop meta-model. *CMES, Computer Modeling in Engineering and Sciences*, Oct. 2009.
4. L. Ehrler and S. Cranefield. Executing agent uml diagrams. In *Autonomous Agent and Multi-Agent System 2004*, pages 906–913, New York, USA, july 2004.
5. J. Ferber and O. Gutknecht. Operational semantics of multi-agent organizations. In N. Jennings and Y. Lespérance, editors, *Intelligent Agents VI. Agent Theories Architectures, and Languages*, volume 1757 of *Lecture Notes in Computer Science*, pages 205–217. Springer Berlin / Heidelberg, 2000.
6. C. Hahn, C. Madrigal-Mora, and K. Fisher. A plateform-independent metamodel for multiagent systems. *Autonomous Agent and Multi-Agent System*, 18(2):239–266, 2009.
7. J. Hubner, O. Boissier, R. Kitio, and A. Ricci. Instrumenting multi-agent organisations with organisational artifacts and agents. *Autonomous Agent and Multi-Agent System*, 20(3):369–400, 2010.
8. M.-P. Huget and J. Odell. Representing agent interaction protocols with agent uml. In *Autonomous Agent and Multi-Agent System 2004*, pages 1244–1245, New York, USA, july 2004.
9. M. Kallmann and D. Thalmann. Modeling objects for interaction tasks. In *Proceedings of Computer Animation and Simulation'98*, pages 73–86, 1998.
10. L. Montealegre Vazquez and F. Lopez y Lopez. An agent-based model for hierarchical organizations. In P. Noriega, J. Vazquez-Salceda, G. Boella, O. Boissier, V. Dignum, N. Fornara, and E. Matson, editors, *Coordination, Organizations, Institutions, and Norms in Agent Systems II*, volume 4386 of *Lecture Notes in Computer Science*, pages 194–211. Springer Berlin / Heidelberg, 2007.
11. A. Omicini and A. Ricci. Mas organization within a coordination infrastructure: Experiments in tucson. In A. Omicini, P. Petta, and J. Pitt, editors, *Engineering Societies in the Agents World*, volume 3071 of *Lecture Notes in Computer Science*, pages 520–520. Springer Berlin / Heidelberg, 2004.
12. H. V. D. Parunak and J. Odell. Representing social structures in UML. In M. J. Wooldridge, G. Weiss, and P. Ciancarini, editors, *Agent Oriented Software Engineering Workshop (AOSE 2001), International Conference on Autonomous Agents*, volume 2222 of *Lecture Notes in Computer Science*, pages 1–16. Springer–Verlag (Berlin), 2002.
13. P. F. Patel-Schneider, P. Hayes, and I. Horrocks. Owl web ontology language semantics and abstract syntax. W3C Recommandation REC-owl-semantics-20040210, W3C, 2004.
14. G. Rimassa. *Runtime Support for Distributed Multi-Agent Systems*. PhD thesis, University of Parma, 2003.
15. V. Torres DaSilva, R. Choren, and C. J.P. De Lucena. A uml based approach for modeling and implementing multi-agent systems. In *Autonomous Agent and Multi-Agent System 2004*, pages 914–921, New York, USA, july 2004.
16. M. Wooldridge, N. Jennings, and D. Kinny. The gaia methodology for agent-oriented analysis and design. *Autonomous Agent and Multi-Agent System*, 3(3):285–312, 2000.