Generation of communicative intentions for virtual agents in an intelligent virtual environment : application to virtual learning environment

Bilal Nakhal

THÈSE / UNIVERSITÉ DE BRETAGNE OCCIDENTALE
sous le sceau de l'Université Bretagne Loire

pour obtenir le titre de
DOCTEUR DE L'UNIVERSITÉ DE BRETAGNE OCCIDENTALE
Mention : Informatique
École Doctorale MATHSTIC

présentée par

Bilal NAKHAL

Préparée au Centre Européen de Réalité Virtuelle
Laboratoire Lab-STICC

–

Generation of communicative intentions for virtual agents in an intelligent virtual environment : application to virtual learning environment

Thèse soutenue le 22 décembre 2017
devant le jury composé de :

Bernard Blandin (rapporteur)
Directeur de Recherches, HDR CESI Montpellier

Paul Richard (rapporteur)
Maître de Conférence, HDR Université du Maine Angers

Laurent Nana (examinateur)
Professeur des universités, Université de Bretagne Occidentale, Brest

Jean Pierre Jessel (examinateur)
Professeur des Universités, Université de Toulouse,

Ronan Querrec (directeur de thèse)
Professeur des Universités, École Nationale D'ingénieurs de Brest,

Elisabetta Bevacqua (encadrant)
Maître de Conférence, École Nationale D'ingénieurs de Brest,

Bilal Said (encadrant)
Maître de Conférence, Arts, Sciences & Technology University in Lebanon

Ali Hamie (invité)
Professeur des Universités, Arts, Sciences & Technology University in Lebanon

ACKNOWLEDGEMENTS

The following companies and schools made this thesis possible:

I would like to gratefully acknowledge various people who have been journeyed with me in recent four years as I have worked on this thesis. First, I owe an enormous debt to my parents and to my wife, Dina, and my three children, Sirine, Omar and Mohamad. Through the struggles and trials of this thesis they have been a constant source of joy.

I would like to express my deepest gratitude to my supervisor Professor Ronan Querrec for his unwavering support, collegiality, and mentorship throughout this project.

I would like to extend my thanks to those who offered collegial guidance and support over the years: Elisabetta Bevacqua, Ali Hamie and Bilal Said.

I would also like to thank my friends, and colleagues at AUL and CERV for their encouragement and moral support.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS AND ACRONYMS

VR Virtual Reality

HMD Head Mounted Display

SME Medium-Sized Enterprises

VLE Virtual Learning Environment

STEVE Soar Training Expert for Virtual Environments

ECA Embodied Conversational Agent

IVE Intelligent Virtual Environment

GVT Generic Virtual Training

SELDON ScEnario and Learning situations adaptation through Dynamic OrchestratioN

#FIVE Framework for Interactive Virtual Environments

MASCARET Multi-Agent System for Collaborative, Adaptive and Realistic Environments for Training

UML Unified Modelling Language

VE Virtual Environment

FIPA Foundation for Intelligent Physical Agents

ACL Agent Communication Language

FIPA-SL Semantic Language

BDI Belief-Desire-Intention

SOAR State, Operator, and Result

ACT-R Adaptive Components of Thought-Rational

LIT Listener Intent Planner

ICT Institute for Creative Technologies

VH Virtual Human

MARC Multimodal Affective and Reactive Character

FML Function Markup Language

BML Behavior Markup Language

SAIBA Situation, Agent, Intention, Behavior and Animation

FML-APML FML-Affective Presentation Markup Language

MURML Multimodal Utterance Representation Markup Language

VHToolkit Virtual Human Toolkit

UDP User Datagram Protocol

SDK Software development kit

AIML Artificial Intelligence Markup Language

DLL Dynamic-link Library

XML eXtensible Markup Language

XMI XML Metadata Interchange

TLX Task Load Index

DV Dependent Variables

IV Independent Variables

ICAI Intelligent Computer-Assisted Instruction

ILE Interactive Learning Environments

AI Artificial Intelligence

LIST OF APPENDICES

1 INTRODUCTION

Virtual Reality is a scientific and technological field exploiting computer science and behavioral interfaces in order to simulate in artificial worlds the behavior of 3D autonomous entities that interact in real time between each other and with one or more immersed users through multiple sensorial channels (such as visual and auditory) [1] [2]. Accordingly, rich multi-modal human interaction input/output systems are used, such as head-mounted displays (HMD), CAVE, motion tracking devices, data gloves and body sensors [2].

Figure 1. Virtual reality head mounted displays

Virtual Reality systems exist since 1980s but they have drawn much attention in the last few years by the general public [3] [4]. As an example, in 2014, Facebook acquired

Oculus VR[1] as one of the first virtual reality headset released for the general public and especially for gamers. Since this date, several vendors like Samsung, HTC, Sony, Microsoft, Google, Epson and LG had also released a large panel of different types of virtual reality HMDs. Among these HMDs, there are low-cost platforms with simple components, such as the Google Cardboard, and advanced platforms, like HTC Vive and PlayStation VR (Figure 1). Earlier, professionals have frequently applied virtual reality and used complex Virtual Reality devices such as the CAVE and the force feedback arms (Figure 2). Even if these devices are useful for industrial and academia and interesting for small and medium-sized enterprises (SME), the future of this type of devices will depend on the success in game development for public.

Figure 2. Complex virtual reality devices

Virtual Reality has been used in several professional domains. Beyond gaming, the entertainment, mental health and shopping fields are considered among the usages of Virtual Reality. Users can be immersed in virtual environments to watch movies, virtually visit historical places around the world, find a relaxing environment to reduce stress and anxiety, or get a shopping experience through a virtual tour of a store [3] [5].

In addition, education is an essential domain applying Virtual Reality [6]. Major virtual reality applications facilitate training in industrial [7], medical [8] [9], engineering [10], sports [11], and military domains [12]. The user can gain more experiences by repeatedly executing the procedure and trying different solutions without serious expenses. Moreover, virtual reality provides users a safe environment to practice and allows them to make mistakes without real impacts. This pragmatic approach of Virtual

[1] http://thefarm51.com/ripress/VR_market_report_2015_The_Farm51.pdf (accessed on November, 2017)

Reality for learning is used by SMEs such as Virtualys and Clarte[2] when developing new products in this field. But Virtual Reality permits developing new educational strategies and this implies new methodologies to create the virtual environment. This is a research domain that we call "Virtual Learning Environment" (VLE) [13]. In the sequel, we delve more in depth in VLEs and the various known approaches to build them. This shall reveal the opportunities left in the literature for us to introduce our thesis contribution.

1.1 Virtual Learning Environment

We propose to divide the competencies that can be acquired in VLE in three categories:

1- Gesture competencies: Users can train in VLEs on gestures of concerned domain. They can perform required tasks in virtual environments using Virtual Reality tools. For instance, a surgical training is applied in [14] using Virtual Reality HMDs with Virtual Reality laparoscopic simulators [15]. Trainers apply surgical activities, such as holding objects with attached blood vessels and separating small vessels from big ones, using real surgical devices that are instrumented in the virtual environment (Figure 3). In this study, the impact of Virtual Reality on the learning performance were time, instrument path length, and angles of used tools.

Figure 3. Virtual reality laparoscopy setup

[2] https://www.virtualys.fr, http://www.clarte-lab.fr/ (accessed November, 2017)

2- Declarative knowledge: Other VLEs are used to acquire declarative knowledge that refers to static information of processes and events with their related attributes, such as teaching soldiers on principles and theories for corrosion prevention and control [16]. VR-based multimedia instructions are presented in the VLE to transmit training information to participants. The Vizard™ VR toolkit[3] is used to construct the virtual environment. Participants could answer the corrosion-based trivia questions by shooting the displayed answers (Figure 4).

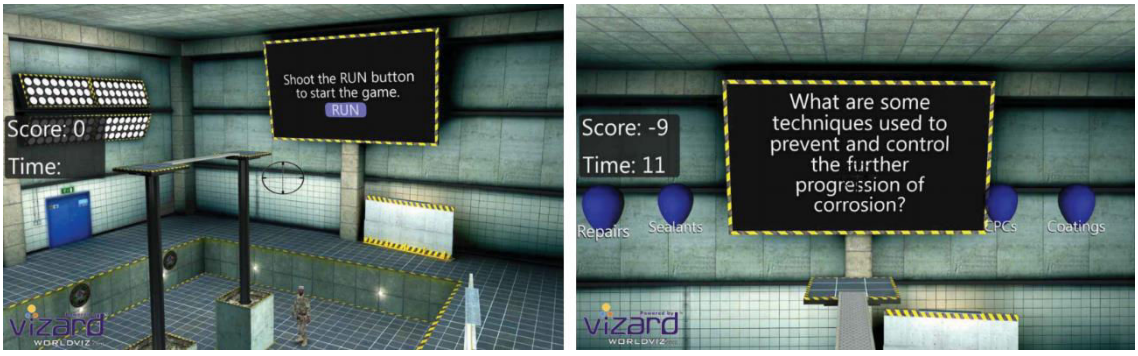

Figure 4. Interactive learning environment for armies

3- Procedural knowledge: A VLE can be constructed to train users on applying procedures. For example, the solution in [12] train engineers on the maintenance tasks of complex engineering system. As shown in Figure 5, trainees have to use a virtual hand to interact with objects in the virtual environment in order to trigger a chain of actions to complete the procedure.

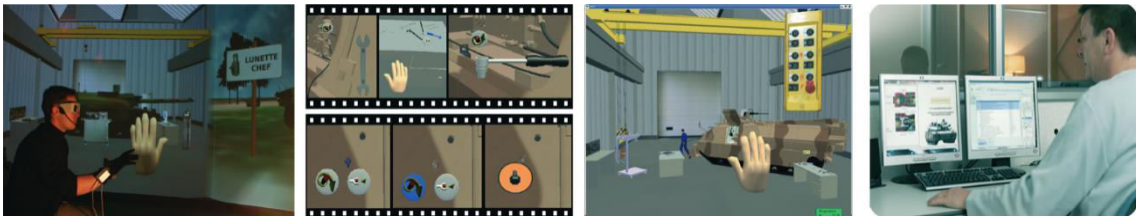

Figure 5. Procedural training on complex engineering system

We can notice how much the domains in VR applications can vary. Such systems can highly cost and consume plenty of time to be developed and updated. Computer scientists have to build the applications based on domain and pedagogical models. What

[3] http://www.worldviz.com/vizard-5-1/ (accessed November, 2017)

is more problematic is that implicitly, the computer scientists may include their own vision of pedagogy while implementing pedagogical scenarios provided by trainers. Therefore, it is crucial in a VLE to consider the methodologies that directly incorporate domain experts and trainers. In this purpose, some methodologies and tools have been already proposed in the literature to build a VLE with the ability for end-users to create or at least to parameterize the domain and the pedagogical scenarios [17]. Significantly, the IMSLD [18], MASCARET [19], #FIVE [20], and SELDON [21] are among the major systems that provide such methodologies.

Furthermore, VLE methodologies can use cognitive science results to propose generic algorithm to generate assistance to the learner to improve his learning performance. These results include the knowledge about several factors for users, like intellectual and emotional, which can support the tutor in building better pedagogical strategies for the learning environments [22]. For example, the proposed system in [17] simulates tutor agents and their cognitive processes to teach users the risk prevention in dangerous working situations.

This type of VLE requires creating an agent architecture with agents that have knowledge bases to hold the domain and pedagogical scenarios along with the information gained from monitoring and interacting with users. Moreover, previous works, like STEVE (Soar Training Expert for Virtual Environments) [23], have proven that virtual learning environment could be improved if they use the same communicative channels that humans use. This means that integrating “Embodied Conversational Agents” (ECAs) can facilitate the interaction between the user and the virtual environment.

1.2 Embodied Conversational Agent

The interaction between users and the VLE may not be so natural and could generate frustration for users [24]. Therefore, a computer interface with human-like embodied conversational agents (ECA) can be used to avoid this frustration. An ECA is a computer-generated intelligent agent that is represented with human-like body. It can interact with other agents and with users within the virtual environment using verbal and nonverbal signals, such as speech, facial expressions and gestures [25]. Experiments have proven that ECAs can motivate users in performing tasks [26].

Originally, only an animated face was representing the ECA (Figure 6-a) [27]. It could show some facial expressions and apply lips synchronization during vocal communication. Afterward, ECAs are represented in virtual environments with 3D human-like embodied agents (Figure 6-b). These agents have to use their body parts (like head, body, arms and legs) to realize physical behaviors and interact with users. The interaction skills of these agents are developed to realize human-like behaviors [28]. ECAs depend on their verbal (speaking) and non-verbal communicative capabilities (such as body gestures and movements) to establish natural interactions with users.

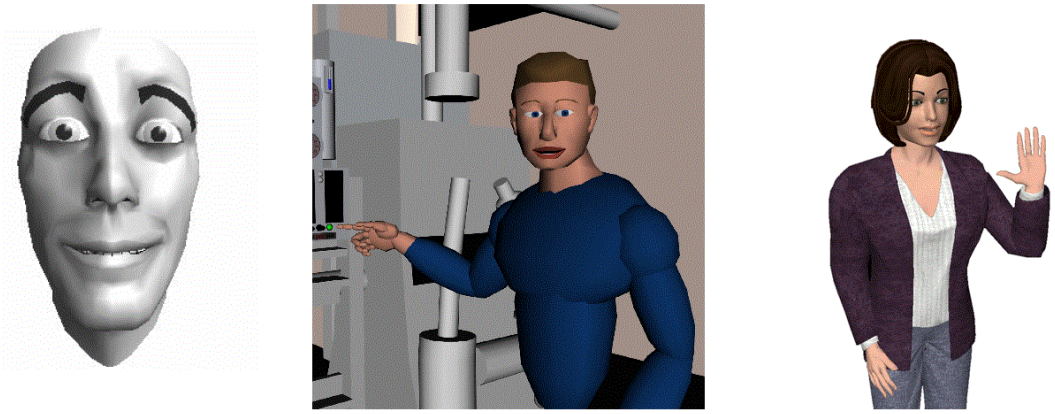

Figure 6. Simple and advanced ECAs

To increase the accuracy of these behaviors, ECA researches are lately focusing on developing these interaction skills so that additional properties, like emotions, can be expressed [25]. Prettifying the facial display of the ECA and adding the social presence for human-computer interactions can make the ECA more realistic and motivate users to confidently interact with virtual environments having ECAs [29].

The researchers of most of the released ECA systems do not include complex knowledge about the environments in the reasoning capabilities of the ECA. They mainly focus on enhancing the physical representation and the communication capabilities of the ECA.

The current research work of ECA systems done by other colleagues, deals more on the social, emotional aspect of the relation between the user and the agent than on the content of the communication. However, in our work, we will ensure that the ECAs have to aware of the context of the virtual environment [30] [31]. They must consider the changing states of entities, located in the environment, that are caused by actions

triggered by the user and other agents [32]. We will work on modelling the knowledge patterns and the reasoning capabilities, defined in the VLE, in order to be able to determine the communicative behaviors of the ECAs by analyzing them [28].

1.3 Thesis objectives and plan

In my thesis, and at a conceptual level, we focus on modelling a VLE that can teach users the procedures of industrial systems (procedure of usage or procedure to repair the system) and that is able to answer to questions they can ask. Compared to classical VLEs, we propose to embody the agents using existing ECA platforms. This permits to motivate users and to have more natural communication abilities. In our proposition, the embodied intelligent agents do not only show natural communication (interaction) with the user but also own a knowledge base on the domain model and the pedagogical model.

At an implementation level, we work on building a real-time system able to sustain natural interaction with the user in a VLE. To implement this model, we use MASCARET as an Intelligent Virtual Environment model [19] [33] that can help giving the knowledge base to the integrated ECA so that it can exchange knowledge with other agents and with the user. Based on this knowledge base, the intentions of the agents are determined and then transmitted to the user through verbal and non-verbal communicative behaviors of integrated ECAs. We aim in this work to formalize the action of communication in MASCARET.

Subsequently, we implemented a concrete application of our proposed model. We built a real life intelligent biomedical tutoring system and applied it as an experimental scenario on several participants. Based on objective performance measures, and similar to previous evaluation works [34], we aim to show the efficiency of our proposed model. We applied the experimental protocols defined by Hoareau et al. [35]. By following these protocols, we can evaluate the interest of virtual reality for learning procedures based on the determined performance measures.

An overview of existing frameworks and tools that could be used to achieve the aimed VLE is discussed in Chapter 2 with the functionalities of the components of the MASCARET meta-model that we developed. In Chapter 3, we present the global architecture of our proposed model. Thereafter, the structure and the implementation details of our application are shown in Chapter 4. In Chapter 5, we detail the results of

applied experiments and evaluation protocols. In the last chapter, we summarize the presented approaches and propose the future perspectives of our work.

2 BACKGROUND AND LITERATURE REVIEW

In this thesis, we aim to build a virtual environment that can be used to provide a knowledge base to agents. This principle is called an "Intelligent Virtual Environment" (IVE) [36], and has been used in several works in the Virtual Learning Environment (VLE) (section 2.1). Having an IVE is not enough to formalize the knowledge base of an agent and its capacity to reason in the goal of naturally interacting with the user. A study of agent architectures is presented in section 2.2. As seen previously, to increase the natural aspect with the user, we propose to use Embodied Conversational Agents (ECA). Some ECA architectures are presented in section 2.3.

In this chapter, we will study the work that is already done in existing frameworks and tools, which we could use to develop an intelligent virtual learning environment.

2.1 Intelligent Virtual Learning Environment (IVLE)

Virtual reality has been already used for educational objectives and had a lot of advantages. The impacts of the considered immersion and interaction modes were evaluated [37] [38]. The main disadvantage obtained in building VR learning applications is that the computer scientists who are developing these applications are intervening in the implementation of all phases of the model including the pedagogical one. They have to design the pedagogical scenarios by applying their understanding of the concerned domain and their own approximation of the learning processes.

One idea is to let the different experts (domain expert and pedagogical expert) to represent them using an external software. In this case, those knowledge are considered as data and can be use in real-time in a virtual environment. This principle is known as intelligent virtual environment (IVE) [36].

IVE models are used to build intelligent virtual reality systems with intelligent operations. It integrates Artificial Intelligence (AI) techniques in the environments of VR systems to enhance the interactivity of a virtual environment. IVE permits building interactive interfaces that are essential to construct the virtual environments of planning and problem-solving applications. Graphical interfaces and natural language interfaces can be considered to represent the knowledge of the system to the user. Based on this knowledge, reasonable behaviors are executed in the virtual environment. IVEs have been used in many applications and research fields. We are focusing in our work to embed an IVE in the architecture of the VLE that we aim to build.

The semantics of the concerned domain, which are addressed by experts and instructors, are represented in the VLE using various knowledge patterns. Accordingly, the autonomous agents, which represent the tutor agents for example, can be assigned in the VLE to interact with each other throughout the virtual environment. These semantics play a fundamental role in managing the methodologies of the learning system in a VLE.

The common requirements of a VLE revolve around building a reactive virtual environment that includes the domain model. In addition, necessary pedagogical strategies have to be defined so that the agents could guide the user acquiring some domain knowledge in the virtual environment (a procedure for example).

Various intelligent technological tools, i.e. pedagogical strategies, which can facilitate applying the learning processes, are integrated in the VLEs [13]. STEVE [23] is among the initial projects that proposed a virtual environment for training. It has a pedagogical agent, called STEVE, which individually helps the user in a virtual environment to train on applying procedural tasks for maintaining a boat.

The generic virtual training (GVT) platform [12] was also established to simplify the development of virtual environments with pedagogical tasks that can help in applying the domain procedures. The produced models of the GVT platform allowed executing more than 50 maintenance scenarios on military equipment.

In fact, to reuse the previously well-established VLEs, such as MASCARET [19], #FIVE [20], and SELDON [21], and benefit from the implemented operations, we have to embed generic models [12] while developing our VLE. Integrating such primary characteristics is a fundamental process to develop a reliable VLE.

2.1.1 SELDON

The SELDON (ScEnario and Learning situations adaptation through Dynamic OrchestratioN) [21] [39] is a dynamic model that focuses on simulating real world aspects in virtual environments (Figure 7). It includes cognitive characters that can naturally react with human-factors. These characters are used by SELDON to control the events while the user is training in the virtual environment. The user has the complete freedom to act in the environment, where the system of SELDON can naturally responds to these actions.

SELDON provides reactive adaptation by managing the consequences of the actions performed by the user. It enables dynamic adaptation by triggering these outcomes in order to assist the user through pedagogical actions. Accordingly, a pedagogical scenario is defined to guide the user in performing required tasks.

Figure 7. Artificial world with real aspects

The SELDON model generates learning situations that corresponds to the activities of the user. It depends on the TAILOR and the DIRECTOR modules that constitute the system architecture of SELDON (Figure 8). The TAILOR module generates learning situations and a sequence of constraints according to the state of the user and the virtual environment [40], and the DIRECTOR module, which is considered as the scenario planner, generates the scenario based on these constraints.

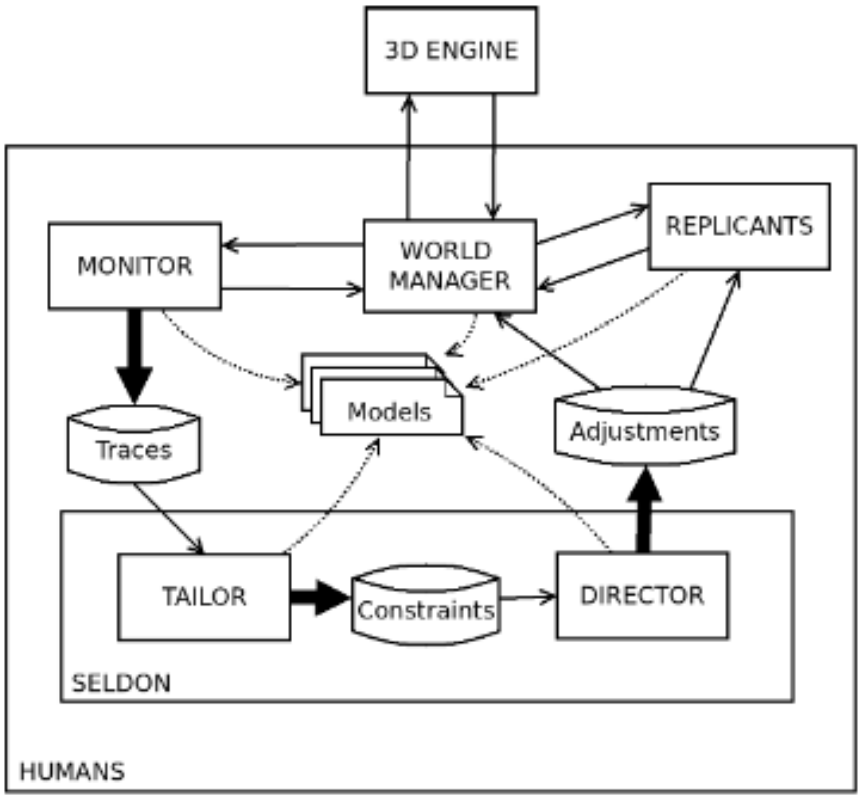

Figure 8. SELDON system architecture

The orchestration in SELDON is a process that determines the desired and feasible scenarios in a learning environment. It monitors these scenarios in order to accurately manage and execute them. The model of SELDON adapts with the running scenario to dynamically control the behaviors. It depends on the constraints of current situations in the scenario that are generated by TAILOR.

While monitoring the user who is working on the training scenario, SELDON considers her/him as one of the virtual characters defined in the virtual environment. It works on expecting the actions of the user and their consequences through the planning process of the model in order to predict upcoming activities.

SELDON is considered as a model that can dynamically generate scenarios in the virtual environment after respecting the activities of the user in real time. These activities updates the beliefs of the system in order to determine the state of the virtual environment and select necessary pedagogical actions.

2.1.2 #FIVE

#FIVE (Framework for Interactive Virtual Environments) is a framework that can be used to build interactive virtual environments. It aims at facilitating the management of the environments in VR applications by:

- simplifying the declaration of the entities' behaviors in the virtual environment
- defining the available collaborative interactive techniques
- building compliant virtual environment to be used on various platforms (Computer, CAVE, Tablet...) and by multiple users
- providing the developer with instructions to ensure constructing an interactive and collaborative VE

Two core modules constitute the model of #FIVE: the Relation Engine and the Interaction Engine (Figure 9). The Relation Engine determines the behaviors of the entities in the virtual environment. However, the Interaction Engine manages the interactions with the user. A communication protocol is also established within these modules to permit the communication between all the modules of #FIVE that can facilitate executing the behaviors (Figure 9).

In addition, these modules permit the developer to manage the behaviors and the relations with the entities in the VE, and to determine the manipulation techniques of these entities.

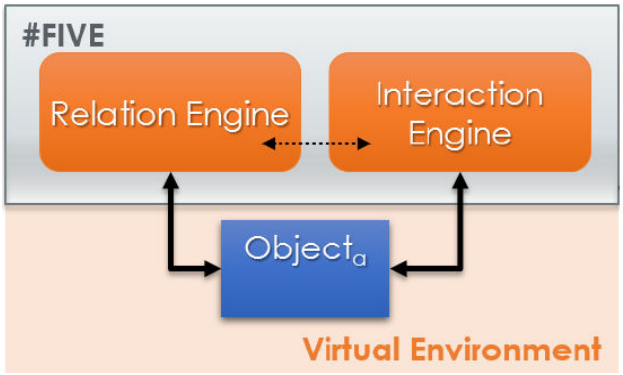

Figure 9. Architecture of #FIVE

#FIVE provides the developer with an abstract layer to construct an interactive, collaborative and distributed VEs. The developer can integrate the #FIVE model in any VE without any restrictions by using any interface engine. The developer has to discuss with the domain expert to get the instructions of the concerned domain in order to code them in an interactive and collaborative environment. Later on, this environment can be distributed to other usages.

#FIVE permits the developer to reuse components from other #FIVE-based application, such as the properties and the behaviors of the entities in the VE. For example, the architecture of the #FIVE-based application in Figure 10 shows how the Relation and

Interaction engines of #FIVE (blue) can simultaneously use the previously defined components of #FIVE-based applications (red) with new components (green) in a new #FIVE application.

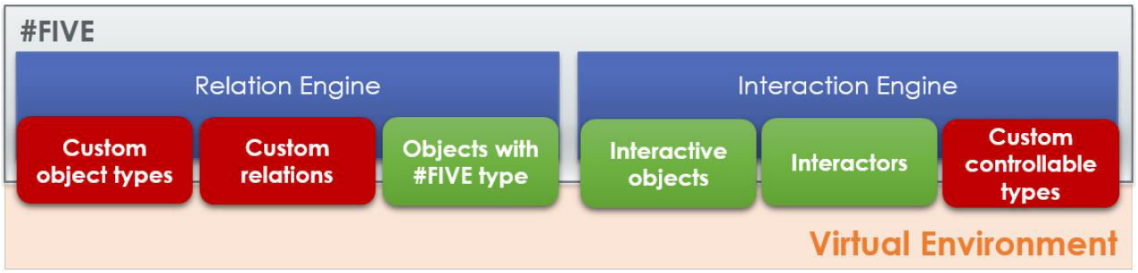

Figure 10. The architecture of a #FIVE-based application

The #FIVE model is still in the development phase. The authors are trying to extend the control of the avatars and to integrate additional interactive and collaborative manners. Furthermore, they are working on considering more independence mechanisms in the framework to distribute #FIVE virtual environments in various 3D interface engines.

2.1.3 MASCARET

MASCARET is a Multi-Agent System for Collaborative, Adaptive and Realistic Environments for Training [19] [33] [41] [42] that can be used to define an intelligent virtual environment. It provides a virtual reality meta-model to describe the semantic of the virtual environment in order for the user and the agents to interact and perform tasks. The structure of the environments, the ontology of the domain, the activities and interactions of the agents and the user, and the knowledge patterns of these agents, are among the main aspects represented in the MASCARET meta-model using the Unified Modelling Language (UML).

In MASCARET, pedagogy is considered as a specific domain model. The same modelling language (UML) is used to describe the domain and the pedagogical model. The pedagogical model is represented by the pedagogical scenario. Koper [15] considers that a pedagogical scenario is composed of five main elements: pedagogical objectives, pedagogical prerequisites, pedagogical activities, pedagogical organizations and pedagogical environments. In Mascaret, pedagogical scenarios are implemented through a chain of actions and activities. Those actions and activities can be either pedagogical actions, like explaining a resource, or domain actions, like manipulating an object.

Class diagrams are used in MASCARET to describe the different types of entities, their properties and the structure of the environment. Asynchronous discreet entity behaviors

are defined through state machines. Activities are designed as predefined collaborative scenarios (called procedures), which represent plans of actions for virtual agents or instructions provided to users for assisting them. The way the activity is interpreted by the agents is defined using specific agent behaviors.

To embed the MASCARET meta-model in a virtual application, the following steps have to be executed:

1- The model of the virtual environment has to be designed by the domain experts using a UML modeller. The experts must use UML diagrams (class diagrams, activity diagrams and state machines) to define the class models, the behavioral models and the actions of the user.

   After finishing the design of the VE using the UML modeller, it has to be exported into an XMI file. Nevertheless, this process should be executed upon modifying or adding any element in the UML diagrams.

2- To construct the virtual environment and the shapes and geometries of objects that should occur in it, developers have to design the scene using 3D entities that are constructed by designers using a 3D modeller. The developers have also to define the behaviors of these entities, and a MASCARET plugin has to be imported to the 3D modeler in order to reference the UML model (XMI file).

3- The user has to launch the simulation platform that loads the virtual environment and all the defined models, such as the domain model and the pedagogical model, and activates the interactive interfaces.

Using MASCARET, end-users (pedagogue, domain expert and domain trainer) are directly involved in the creation of the VE (as seen in Figure 11). They are responsible for constructing all the elements of the virtual environment for learning (domain scenario, pedagogical scenario…).

The pedagogue defines the pedagogical scenario to assist the user in performing the domain application in the VE. This scenario consists of a sequence of pedagogical actions that should be linked to the domain actions that the user should perform and to the objects of the VE that should be manipulated. The domain experts, who define the activity that the user should learn, formalizes the sequence of actions and interactions with the objects of the environment.

The domain trainer defines pedagogical scenarios (the sequence of situations in which the trainee acts in the environment) and the pedagogical assistance provided by the

system in real time. To define the scenarios, the domain trainer uses (1) the environment and the objects it contains, (2) the potential actions of the user on the objects and the good practices (defined by the domain expert), and (3) the generic pedagogical actions (defined by the pedagogue).

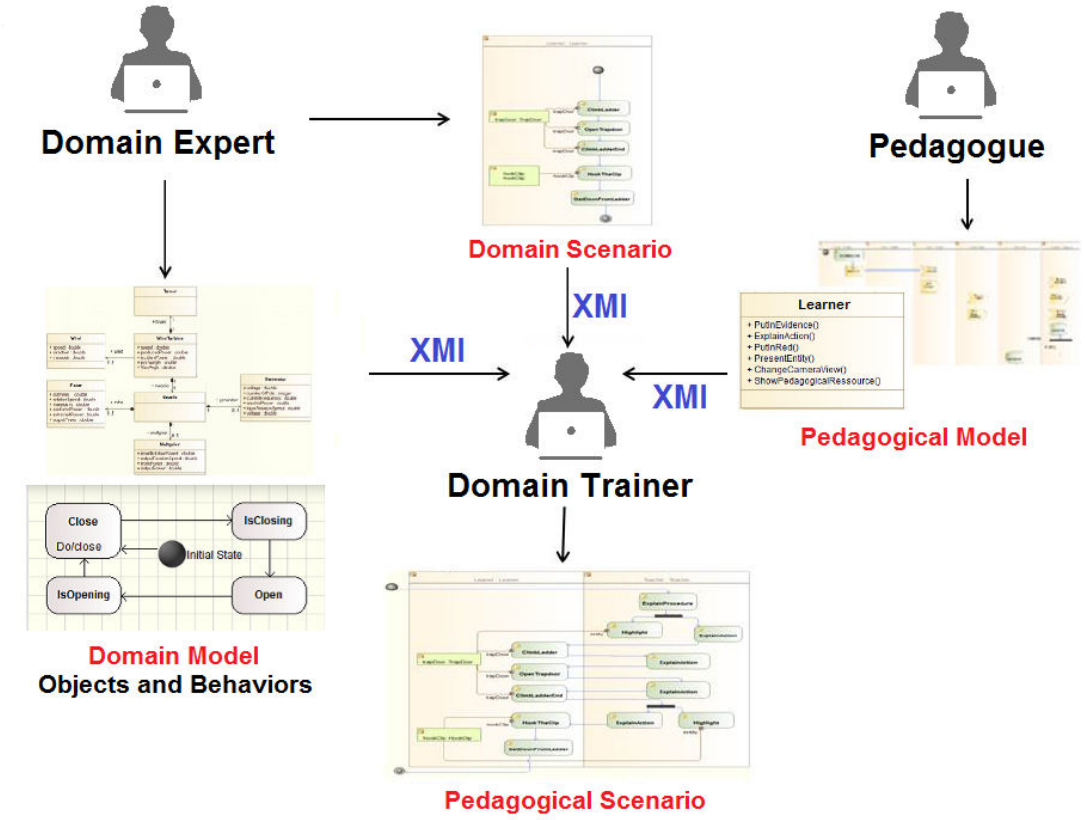

Figure 11. Mascaret workflow to design a virtual environment for learning

2.1.3.1 The agent metamodel

The same concepts for defining the models, using the UML, are used in MASCARET to build the meta-model of the multi-agent system to simulate the activities of the user in the environment. These activities require manipulating objects in the virtual environment. Therefore, MASCARET uses the same language (UML) to define the activities and the environment. Several agent meta-models were proposed in the literature and used UML but did not incorporate modelling the environment with the activities.

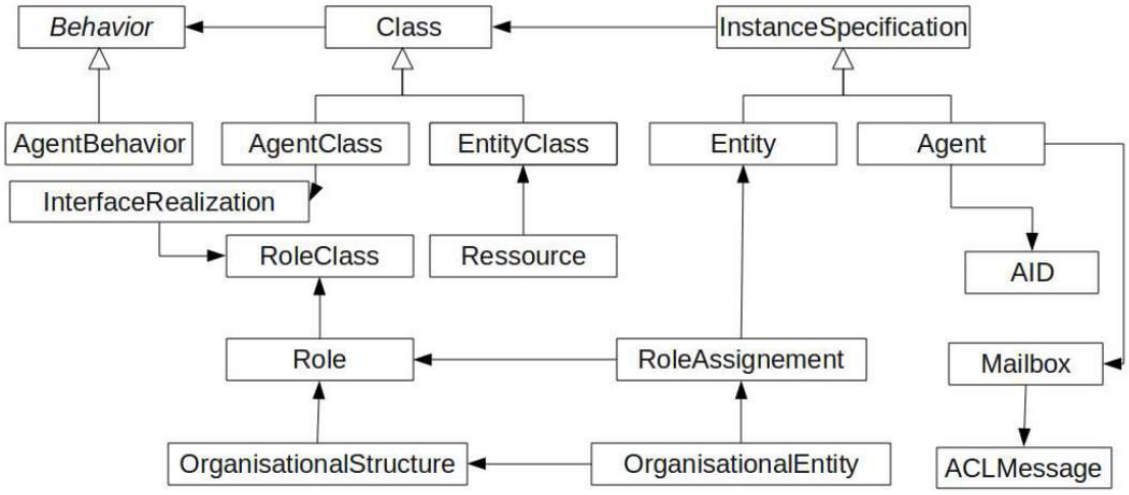

Figure 12. The agent meta-model of MASCARET

Figure 12 represents the concepts that are involved in building the agent meta-model of MASCARET:

- *Multi-Agent System and behaviors of the agents:* The agent is an instance in the environment that is characterized with several properties and actions that are actually formulated in the **AgentClass** (Figure 12). It can perform behaviors and communicate with other agents. MASCARET provides a multi-agent system to define several agents that can execute several behaviors, such as the pedagogical behavior. Each agent has a name and is hosted by an agent-platform. Any property, action, behavior or even an agent in MASCARET can be easily added or modified without affecting other elements in the model.

  MASCARET implements the model of JADE [43] to describe the behaviors of the agents. The behaviors consist of a set of actions that are arranged in a procedure and scheduled by MASCARET.
- *Communication of agents:* A standard communication protocol is considered to facilitate the communication between the agents. MASCARET uses the FIPA (Foundation for Intelligent Physical Agents) protocol that adopts the FIPA-ACL (Agent Communication Language) specifications. Each agent in MASCARET has a communication behavior that is responsible for managing and analyzing the FIPA-ACL messages that are exchanged with other agents. The purpose of the FIPA-ACL message is represented by one of the 23 performative functions proposed by the FIPA. The agents of MASCARET uses the REQUEST performative to obtain the value of a property or to make it execute an action. In response, these agents use the INFORM performative to inform about the value

of the specified property or to confirm executing an action. The researchers and developers of MASCARET are still working on using the other performatives to evolve the communication aspects of the agents.

- *Organization:* To endow modularity to the behaviors of the agents, MASCARET defines an organization according to the specific rules of the behaviors and the agents. Based on the structure of the environment and the defined agents, the organizational entity is created to specify the rights and the duties of these agents by structuring their roles using the **RoleClass** (Figure 12). A role describes the responsibility of the agent and determines the actions it can execute. An agent could be prevented from applying certain actions that another agent is responsible to execute.

  The organizational entities of MASCARET are defined in UML since it depends on the properties of the environment and the agents which are also defined in UML. Therefore, the behaviors of the multi-agent system are described using an activity diagram (considered as an organization) to ensure implementing the rules of the behaviors and defining the role of each agent. The organizations and the roles are linked to the environment in order to be able to use its entities (resources) that are defined using the **Entity** class.

  An XML file is dynamically managed in MASCARET to instantiate the entities of the environment, the agents and the organizational entities.

2.1.3.2 Pedagogical Scenario

The adaptability of the VLE ensures executing behaviors depending on the context of the environment and on the effectiveness of activities performed by the user. For this reason, it is not typical to statically assist the user through a learning scenario. A pedagogical scenario with appropriate pedagogical activities has to be defined and linked to the requested activities and their relative properties [18].

The pedagogical scenario is an organization for learning activities. It is implemented in MASCARET and includes several properties: the objectives of the domain scenario, a list with the prerequisites of actions, the virtual environment, the activities of the pedagogical scenario, and the pedagogical resources that are linked to the entities of the environment that should be manipulated by the user to perform required actions. The agents could reason on the properties of a pedagogical scenario which are considered as its knowledge base.

Several tutoring models with pedagogical scenarios are previously built in VR, like FORMID [44]. The learning behaviors of the pedagogical scenarios in these models depend on the continuously estimated states of manipulated objects but are previously chosen at the design stage. Such models miss some significant characteristics like genericity and do not separate between the procedural scenarios and the pedagogical ones.

Figure 13 shows the activity diagram of an application that integrates the MASCARET meta-model to define two roles: the domain actions for the trainee, and the pedagogical actions for the virtual teacher. The user does not know *a priori* how to perform the domain actions and which objects to manipulate. Therefore, a pedagogical scenario that includes all necessary properties is appended. We can view in Figure 13 how the pedagogical actions are linked to the domain actions and to the entities. This procedure starts with an action to explain the objective of the domain scenario that the user should attain.

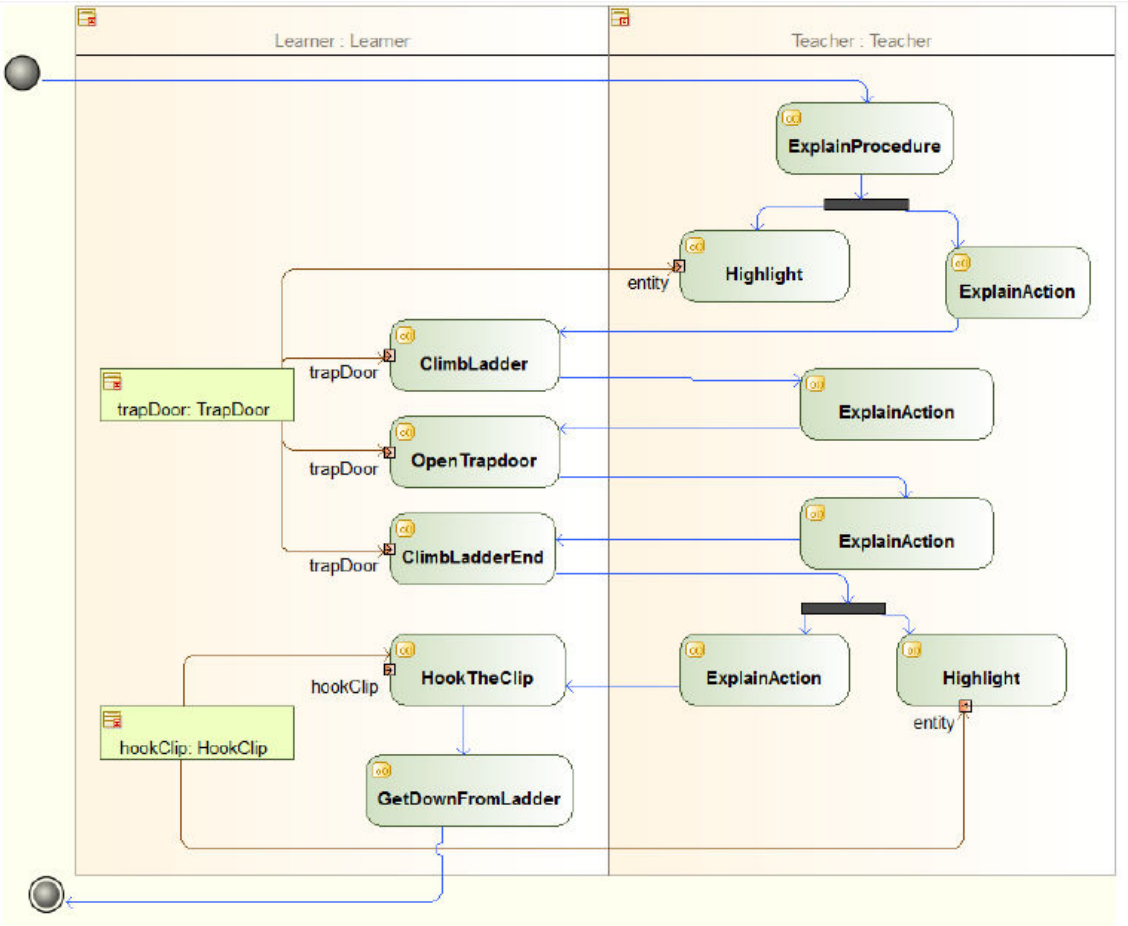

Figure 13. UML activity diagram of a MASCARET application with a domain model for the trainee and a linked pedagogical model for the virtual teacher

The domain teacher is capable of updating any element in this generic UML activity diagram, such as adding/removing a pedagogical action or even changing the description of this action. To consider these updates, the XMI file has to be exported from the UML modeller upon each update.

While executing the scenarios, the virtual agents monitor the activities of the user to provide necessary assistance when required. The agent in MASCARET depends on the semantic modelling of the virtual environment and the activities [33].

In the following section, we discuss the most significant cognitive agent architectures, BDI, SOAR and ACT-R, and their main functionalities that could be embedded in the structure of the IVLE.

2.2 Cognitive architectures

Intelligent systems (like autonomous agents and intelligent tutoring systems) can recognize, reason, learn, and act intelligently using the accessible knowledge in intelligent virtual environments. The infrastructure of such systems can be defined using a cognitive architecture which applies various human intelligent aspects (cognitive functions).

Cognitive architectures are produced to build the structure of intelligent systems. It represents the knowledge of these systems, which is composed of the contents of the concerned domain and the acquired knowledge of performed activities in the environment [45] [46]. Most of the intelligent tutoring systems integrate cognitive architectures in their structures in order to provide accurate explanations to the user in a virtual learning environment.

An agent with a cognitive architecture can consider all the knowledge patterns of the concerned domain, the pedagogical scenarios, the actual context of the environment and the impacts of user's activities [47] [48]. Several modules that represent these knowledge patterns collaborate to include the history of attained processes and the altered states of environmental entities. The user starts executing the procedural scenario without knowing the steps that should be followed. The cognitive resources can be used to direct the user for applying the desired actions and manipulating the proper objects. When the user gains the ability to perform the procedural scenario, the procedural knowledge is considered to be acquired and less cognitive requests are needed [49].

It is essential to embed a significant cognitive architecture when building a VLE, so that the roles of its virtual agents can be promoted. According to [50], a cognitive architecture provides the agents with knowledge patterns and reasoning capabilities, namely:

- **Semantic knowledge**: The knowledge which is known a *priori* by the agent, such as the knowledge about the plan of actions and entities in the virtual environment.
- **Perception to knowledge modifications**: Acquiring new or updated states of entities in the virtual environment.
- **Cooperation of agents**: The communication between the agents while cooperating to achieve individual or collective goals.
- **Planning**: Planning for an arranged sequence of actions with their expected impacts and consequences.
- **Reasoning**: Considering all the facts that could influence forming the sequence of actions.

Based on such capabilities, the intelligent behaviors can autonomously be prepared and handled by the virtual agents of the VLE. Moreover, the agents in the VLE cooperate to perform the assigned activities by sharing their knowledge bases. They communicate with each other by exchanging messages built upon an agent communication language.

2.2.1 Significant cognitive agent architectures

Different models and architectures, like BDI (Belief-Desire-Intention), SOAR (State, Operator, and Result) and ACT-R (Adaptive Components of Thought-Rational) are developed to build cognitive frameworks for intelligent agents in order to be able to plan for human-like natural behaviors across different domains and applications.

The Belief-Desire-Intention (BDI) architecture [51] [52] [53] is one of the best approaches that are considered to build an intelligent system for tutor agents in order to illustrate human reasoning models. It assigns short and long term memories among the knowledge base of these agents. The short term memory in BDI includes the beliefs and facts about the virtual environment [54]. It uses a database to save the context of the virtual environment with the continuously updated states of its virtual entities. Emotional [55] and social states of agents, forming its internal state, are also estimated

according to obtained context. Therefore, all patterns of knowledge are accordingly modified upon the perception of any activity.

Every agent holds a long term memory [54] that holds the plan of goals and desires it has to attain, such as the tutoring and the pedagogical objectives. The agent depends on its beliefs in order for suitable intentions to be selected. Behaviors and actions are then assigned to the agent to achieve its determined desires.

The beliefs, desires and intentions are considered as the mental attitudes carried by the BDI agents which reactively cooperate to achieve targeted objectives (Figure 14). The BDI architecture focuses on constructing the intentions of the agents to represent their commitments and execute particular plans of actions [56].

Several specifications of BDI architectures have been already released, such as formal specifications using standard software engineering tools [57] and procedural concepts for building procedural reasoning systems. These BDI architectures are still the most durable agent architectures that are being used [58].

Researchers in [59] improved the BDI architecture by relating the representation of knowledge in the framework to the expressed knowledge of experts through particular learning techniques. Consequently, several agent-based systems, like Jadex [60], adopted the BDI-model for the implementation of intelligent agents.

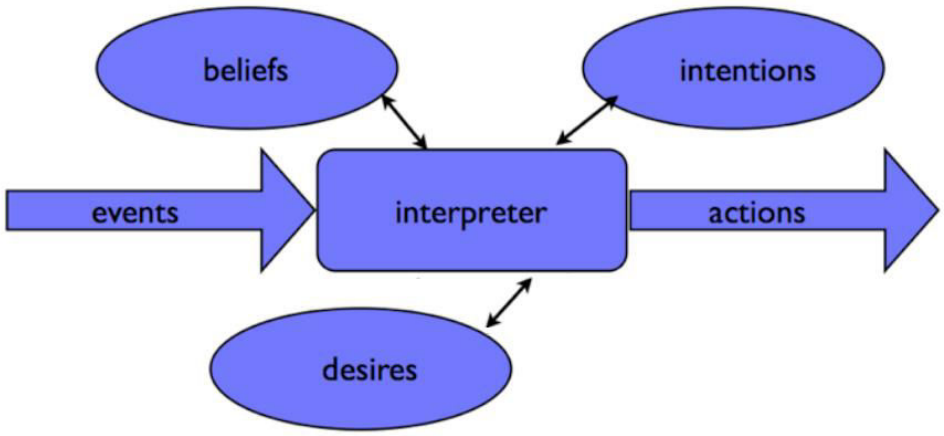

Figure 14. The BDI architecture

The State, Operator, and Result (SOAR) cognitive architecture [61] focuses on building the agent's intelligence according to its acquired experience. Recent SOAR evolutions cover human-like behaviors by appending educational mechanisms, long term memory and various types of knowledge (Figure 15). These mechanisms have a major role to develop reasoning, planning and decision making processes in the VLE.

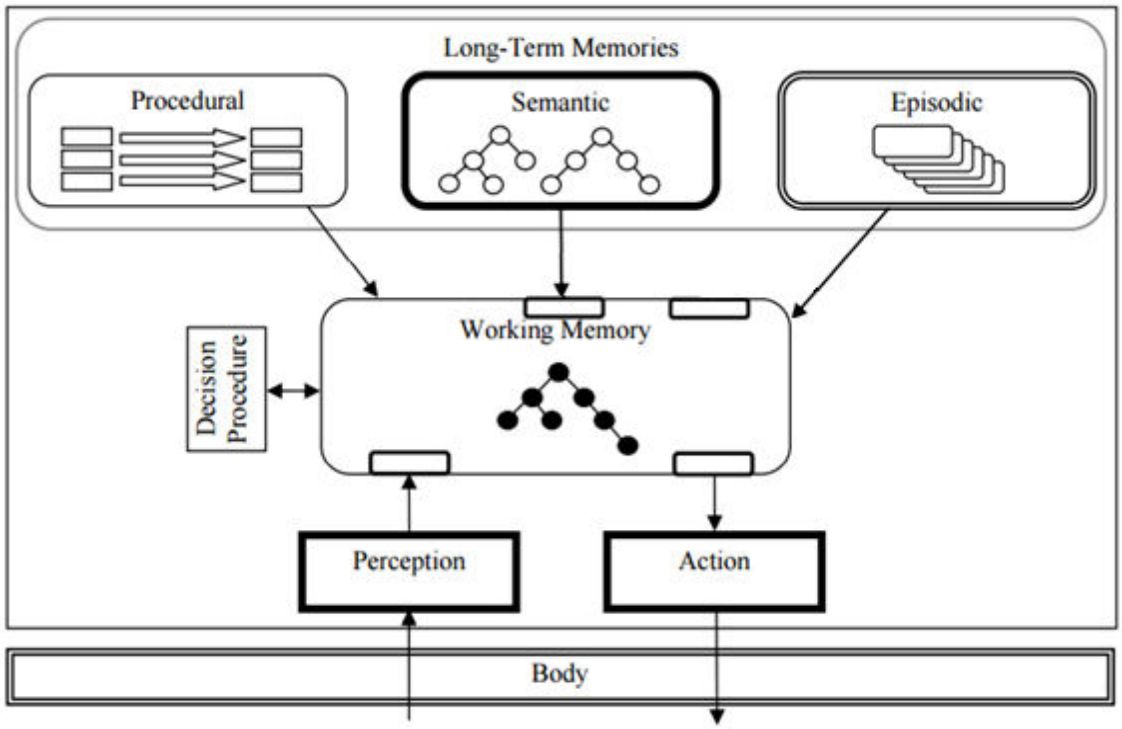

Figure 15. Memory structures in Soar [62]

The long term memory holds the procedural knowledge that handles the most applicable behaviors of the concerned model, while the semantic memory holds the context of the virtual environment. However, the working memory is considered as the short term memory responsible for processing the current event the agent is working on. The proceedings of modified states are saved inside the episodic memory which is recently altered by [63]. When the required states, which are set in the working memory, are obtained within the environment, the agents can then realize essential behaviors.

SOAR and BDI provide quite similar reasoning capabilities to the agents [64]. They both require a knowledge base to store various facts and inputs from the context of the virtual environment. They use a decision-making procedure to generate intentions based on the desired output or goal. However, SOAR considers the elementary actions, but not plans, i.e. sequences of actions, and does not offer flexible means to define the preference semantics of the reasoning phase that generates the intentions of agents [64].

The ACT-R (Adaptive Components of Thought-Rational) [65] is a cognitive architecture that applies a modular decomposition of cognition for modeling human behaviors [66] [48]. It proposes a theory to integrate these modules in order to build reasonable cognition (Figure 16). Among these modules, we have the perceptual module for recognizing visual entities in the environments, the motor module to manage

and apply actions, the declarative module to get information from the long-term memory, the goal module to follow the steps to solve assigned tasks, and the production module that organizes the tasks of all modules. The production module cyclically demands information from other modules using the operations of the system. Certain constraints are requested by sending and receiving chunks through buffers.

The ACT-R system can only launch one production to retrieve the knowledge of a certain memory, while with SOAR several productions could be simultaneously launched.

The architecture of ACT-R has been embedded in several psychological studies and VLEs, such as the shared work with fRMI data [67]. However, ambitious strategies for problem solving and reasoning are still needed.

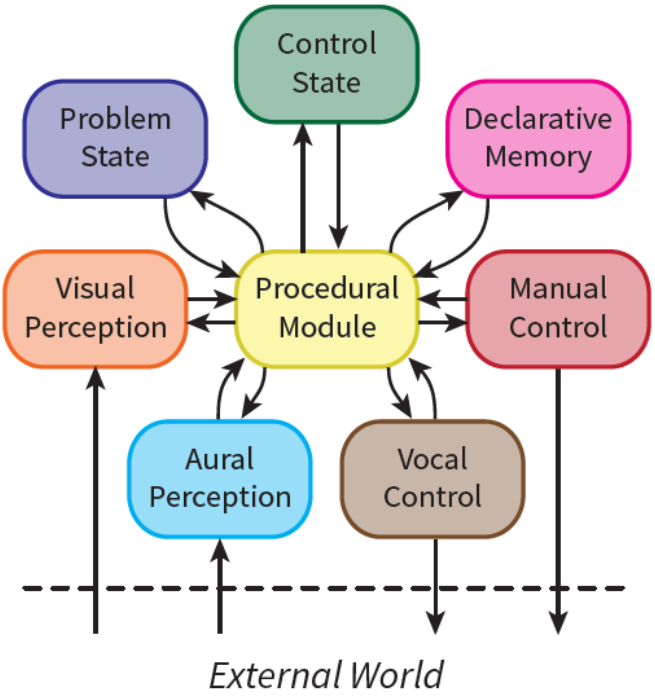

Figure 16. The main modules of ACT-R [67]

2.3 Embodied Conversational Agent (ECA)

The Embodied Conversational Agent (ECA) is a computer interface that is represented as a human-embodied agent that can naturally interact with the user. Verbal and non-verbal behaviors can be realized by an ECA including vocal speech, facial expressions, hand gestures and other body movements. These behaviors allow the ECA to communicate with the user in the most human-like methods which could motivate the user to respond and interact [68].

The embodied agents have to be characterized with several capabilities to reach the level of human intelligence throughout their interactions with the user, such as planning, and emotional reasoning [69]. However, the influence of using an ECA for interacting with the user can be evaluated by reviewing their responses and comparing the results of performed scenarios [68].

2.3.1 Interest of ECA for virtual learning environment

ECAs are progressively being developed to adopt the most realistic human visual representation and communication capabilities [70]. They are considered as computer interfaces that could replace human tutors, for example, in practicing and learning scenarios. During the interaction with the user and the virtual environments, these agents have the ability to execute verbal and non-verbal behaviors like speaking, facial expressions, body gestures and locomotion activities. Experiments, like in [26], prove that involving ECAs in learning scenarios as tutor agents can motivate the user to accomplish required tasks.

The capabilities of an ECA can draw the attention of the user with the most common and natural manners, such as gaze and deictic gestures. For instance, it would be considerable if the ECA looked at an object and pointed to it while moving in the virtual environment and discussing the required task with the user. Such human-like manners are essential to provide the user with realistic contexts and motivate her/him to naturally interact with the ECA and consider the ECA as a human tutor.

Several ECA projects have been developed so far, but ECA with light and specific domain knowledge, like STEVE [23] and MAX [71], were initially created. Latest ECA researches are further focusing on achieving more credible intelligent ECAs by improving the natural interaction (human-like), the intelligent capabilities, the emotions and the facial expressions. The Virtual Human Toolkit[4] [72], Greta[5] [73] and MARC[6] [55] are examples of the currently utilized ECA platforms.

[4] https://vhtoolkit.ict.usc.edu/ (accessed November, 2017)

[5] https://perso.telecom-paristech.fr/~pelachau/Greta/ (accessed November, 2017)

[6] http://www.marc-toolkit.net/ (accessed November, 2017)

2.3.2 ECA platforms

Among the initially established ECA platforms, STEVE (Soar Training Expert for Virtual Environments) [23] was used as a tutor agent to execute particular pedagogical scenarios. Using SOAR, STEVE was developed to interact and train the user on predefined operations of a ship's control panel. STEVE is an interactive system since it interprets various input sources from the user such as keyboard strokes, mouse clicks, and voice commands. STEVE interacts with the user through an animated embodied agent that has a physical representation of a human-like face and body. It uses gestures to communicate while navigating in the virtual environment.

In consequence, the primary tasks of STEVE revolve on demonstrating required actions to the user and on observing the performed actions. STEVE can support the user when needed by replying to inquiries about prior actions.

The concerned domain knowledge of STEVE includes the initial states of entities in the virtual environment, and the procedural scenario that should be followed. Nevertheless, STEVE is characterized with several human-like capabilities which weren't provided by previously utilized agents. It can realize several human-like actions and movements, reply to inquired questions, use gestures and gaze actions, follow implementing the sequence of actions of assigned procedure, and uses its memory components to record performed actions and altered states of the entities. For this purpose, several components are supplied by the architecture of STEVE:

1- Simulator: applies the behaviors in the virtual environment
2- Visual Interface: permits the user to interact with the virtual objects
3- Audio Component: needed to vocalize and to accept vocal messages from the user
4- Speech Generation: transforms the created text messages into speech in order to be vocally transmitted to the user
5- Speech Recognition: receives vocal messages from the user and transforms them into natural language text in order to be analyzed by the system
6- Agent: represented by an ECA which is responsible to naturally realize the selected behaviors

Moreover, the architecture of STEVE is composed of the three main modules [74]: perception, cognition and motor control (Figure 17). STEVE uses these modules simultaneously in order to execute desired scenarios. The perception module recognizes

the consequences of actions performed by the user and the agent and records the altered states of entities. The cognitive module analyzes the recognized input stream and relatively reviews the domain knowledge components in order to plan for the objectives to be achieved. Appropriate actions and motor commands are then specified and sent to the ECA in order to be realize.

STEVE uses a stack to manage the planned tasks of a given scenario. The sequence of actions is set in this stack in order to progressively be executed. To perform an action, STEVE checks for the position of the object that should be manipulated, and then moves to its location. Before applying the action, STEVE points to the object and describes the action that should be performed. For example, as in Figure 18, STEVE is pointing to the power light and vocally explaining the action that should be performed. After demonstrating the action, obtained results are shown.

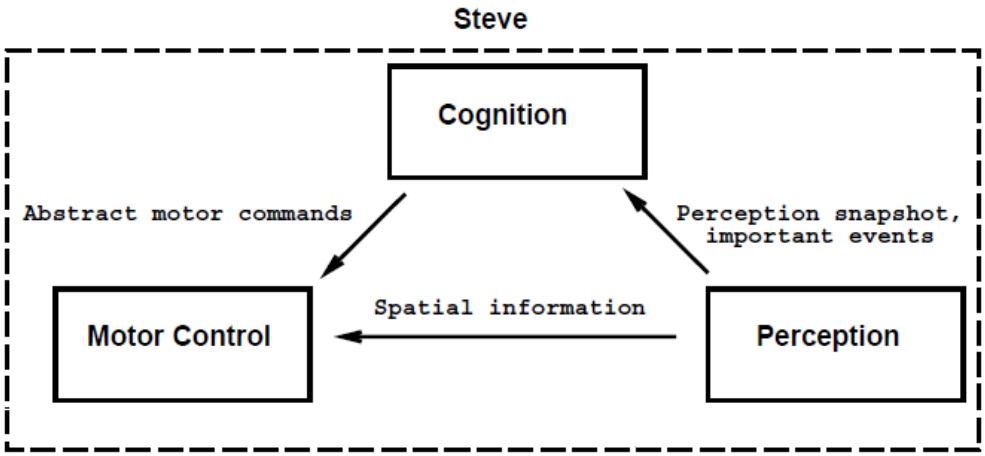

Figure 17. The three main modules of STEVE

As a matter of fact, actions and inquiries of the user are continuously being monitored by STEVE. When a performed action is recognized, STEVE compares it with the current required action in the procedure. If the performed action is valid, obtained results are explained to the user and the following action is stated. While if an inappropriate action is detected, STEVE rejects the action and re-explain the desired one.

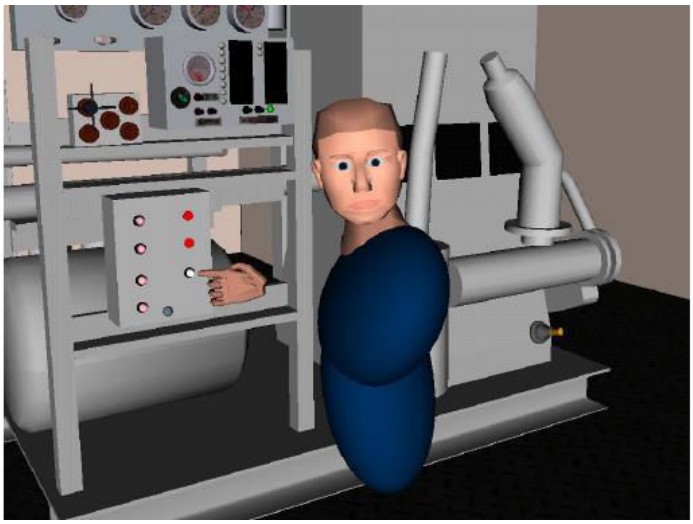

Figure 18. Steve describing the actions to be performed

The user can direct various questions to inquire about the required actions and the environmental objects to be manipulated. For instance, the user can ask about the next action that should be performed (User: “What should I do next?”), STEVE has then to inform the user with the specifications of the required action. The user might fail to perform this action, and asks STEVE to show its appliance (User: “Show me what to do”), STEVE has then to demonstrate the action in the virtual environment.

After all, STEVE is just a mono-agent system that uses a virtual agent to act as a tutor. In certain scenarios, having several agents with different roles can be a major requirement to implement required solutions. For this reason, it is necessary to use a multi-agent system where further operations could be involved.

Besides, the pedagogical strategies defined in STEVE are constant. It is preferable to be dynamically considered according to the acquired knowledge bases of the agents. For this reason, the learning environments must be flexible and adaptable where several intelligent capabilities have to be appended to the utilized ECA platforms. ECAs like Greta, MARC and the Virtual Human Toolkit can be integrated in such environments since they differ in complexity, graphical output and application domains.

2.3.2.1 Greta

Greta [75] [76] is an ECA platform that focuses on two main properties, believability and individuality. These properties enhance the interaction skills of embodied agents represented in the virtual environments in order to naturally interact with the user by realizing verbal and non-verbal communication behaviors. In Greta, the context of the virtual environment is also considered to apply synchronized multimodal behaviors.

This can increase the believability of the agent and motivates the user to communicate with the ECA.

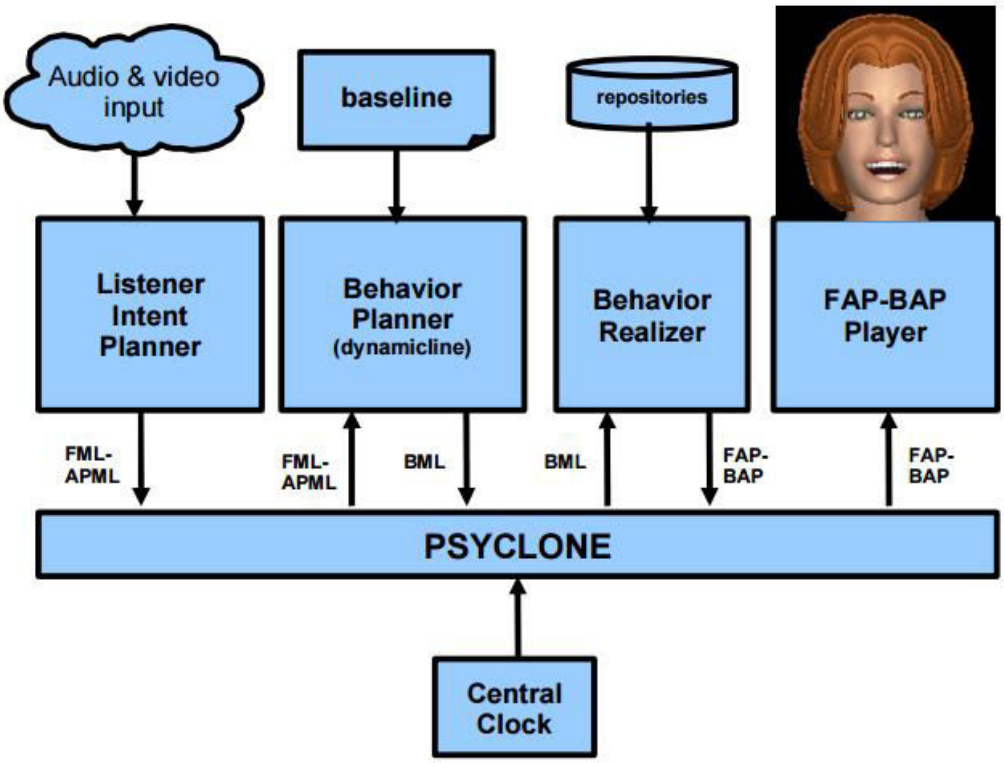

Figure 19. The architecture of Greta

Furthermore, the inner states of the ECA are also considered in Greta. The real feelings and the individualized properties of each ECA, such as emotions, culture, mood, gender, and age, can be set. The modular structure, functionalities and the communication protocol of the ECA are defined in the architecture of Greta using the SAIBA framework [77]. SAIBA, which is detailed in the following section, is responsible for representing the communicative intentions and the communicative behaviors that the ECA has to realize using the two standard XML languages FML and BML.

Among the architecture of Greta (Figure 19), the Listener Intent Planner (LIT) receives the input from the user including audio and visual information, such as vocal messages and head nods. The communicative intentions of the agent are then chosen and represented in the FML-APML script. It is defined by the intent planner (first module in SAIBA) using an input file, where the communicative intentions of the ECA are determined and sent to the behavior planner (second module in SAIBA) via the Psyclone messaging system [78]. After that, the behavior planner generates the corresponding communicative behaviors and represents them using a BML script. The signals of these behaviors are sent to the behavior realizer (third module in SAIBA) to

produce MPEG4 FAP-BAP animation files in order to display the realized behaviors on the ECA using the FAP-BAP player [79]. In addition to the FML-APML input stream, the mentioned discrete high-level inner states and the set of expressivity parameters are linked together to build MPEG4 animations that can demonstrate the behaviors of an individual ECA in the platform of Greta (Figure 20).

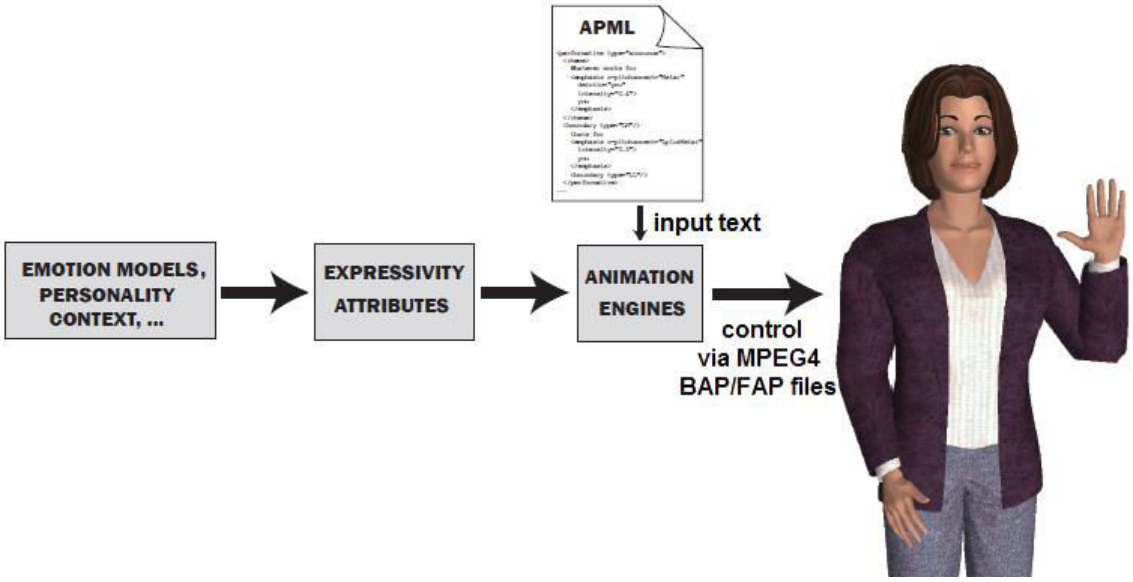

Figure 20. The individualized actions of Greta

2.3.2.2 Virtual Human Toolkit

The project of the Virtual Human is constructed at the Institute for Creative Technologies (ICT) [80]. Its objective is to build and naturally structure embodied agents that can realize human-like actions while interacting with the user during the implementation of social trainings in the virtual environments (Figure 21).

Figure 21. Virtual Humans

In fact, Virtual Humans (VH) are autonomous agents that perceive their environments and recognize performed activities in order to accordingly update their beliefs. They model their own and other's believes, desires and intentions, and follow the assigned plans to naturally interact with the user by realizing verbal and non-verbal communication behaviors. Furthermore, various roles can be handled by these agents for supporting the user in executing training scenarios [80].

To naturally collaborate with the user and other cooperating agents, several capabilities are carried out by the VH agents [69], such as the automated speech recognition, perception using the Computer Expression Recognition Toolbox (CERT) [81], task modeling using the DTask [82], natural language generation [83], and the text-to-speech using the Festival engine [84].

Consequently, the ICT developed the architecture (Figure 22) of the Virtual Human Toolkit that defines, at an abstract level, the essential modules that can properly realize the functionalities of the virtual human. The VH Toolkit is composed of several modules, tools, libraries and 3rd party software that cooperate to attain these functionalities.

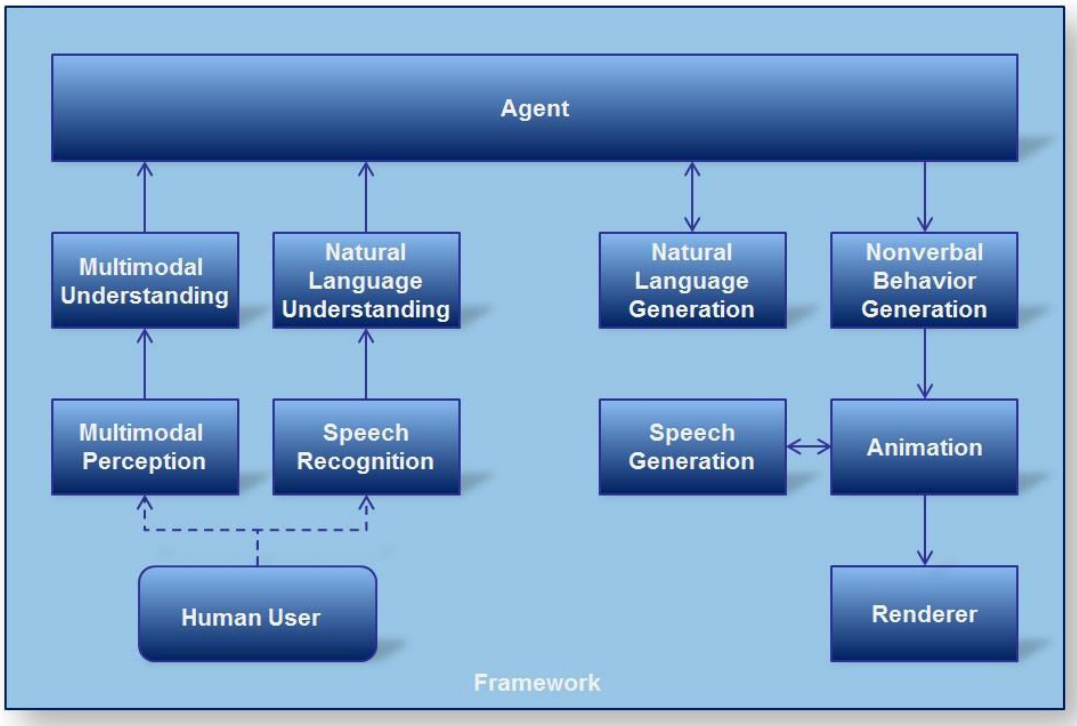

Figure 22. The Virtual Human Architecture

When the user verbally interacts with the system, the speech recognition module transforms the speech into a natural language text that can be used for reasoning by the dialogue manager of the agent. However, the audio-visual sensing relies on sensory input that can also recognize the nonverbal communication behaviors.

After analyzing these inputs and reviewing the internal state, the VH works on building the communicative intents in order to properly reply to the user. Based on these intents, suitable verbal and non-verbal behaviors are generated. The behavior realization module of the VH synchronizes the realization of all behaviors such as the speech, lip synching and facial expressions.

2.3.2.2.1 Lists of questions and answers

Among the integrated modules and tools within the VH Toolkit architecture, and upon the creation of a VH character, the NPCEditor [85] is used. The NPCEditor is a text classifier that acquire the speech of the VH characters that corresponds to the questions asked by the user. It depends on a dialogue manager that contains a list of questions that the user might ask while interacting with the VH, and linked to a list of answers that the VH characters have to verbally realize among the communication behavior [86].

The NPCEditor includes a .plist file that contains the lists of questions and answers and their essential properties such as ID, Text, Speaker, Type, Domain and Score. These properties are used to properly link answers to expected questions listed (as shown in Figure 23). The dialogue manager can be modified in the NPCEditor to specify new lists and update the properties of the linked questions and answers [86].

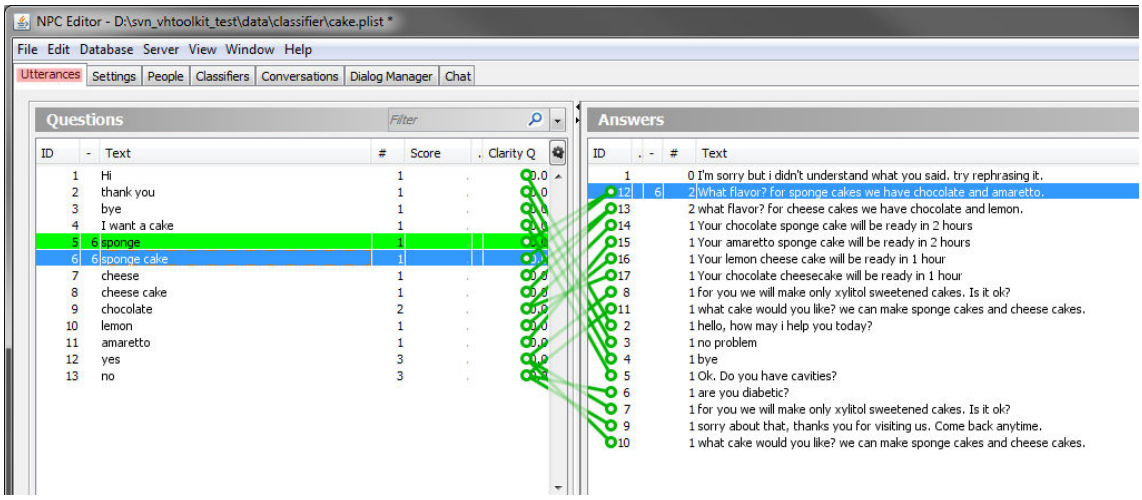

Figure 23. Linking questions and answers using the NPC Editor

Although the questions can be easily managed by the NPCEditor, still it is not considered really dynamic or generic when all possible questions have to be listed. For example, if we have three objects (**OBJ1**, **OBJ2** and **OBJ3**) in the virtual environment, we have then to include all possible questions about them, such as:

- What is **OBJ1**?
- What is **OBJ2**?
- What is **OBJ3**?

Alternatively, we propose a solution to this problematic in our model (in Chapter 4) where generic questions can be adopted.

2.3.2.3 MARC

MARC is a Multimodal Affective and Reactive Character that is created to examine the influences of expressions of virtual agents when interacting with the user among affective computing applications, and their abilities to appropriately respond with real-time affective behaviors and expressions (Figure 24) [87].

Figure 24. An affective computing application of MARC

The real-time interactions and the emotional models are frequently limited in the previously developed virtual agent frameworks. However, in MARC categorical, dimensional, cognitive and social emotional approaches are implemented. The framework of MARC allows examining various emotional models and considering multimodal interactions with expressive virtual agents [55].

MARC can be integrated in any suitable environment such as a VLE. The tools found in such environments can communicate with MARC to manage the behaviors that he should realize in real-time by sending messages using the Behavior Markup Language (BML). Moreover, a toolkit with a set of tools that can manage the characters of MARC and their properties is lately released (Figure 25) [88].

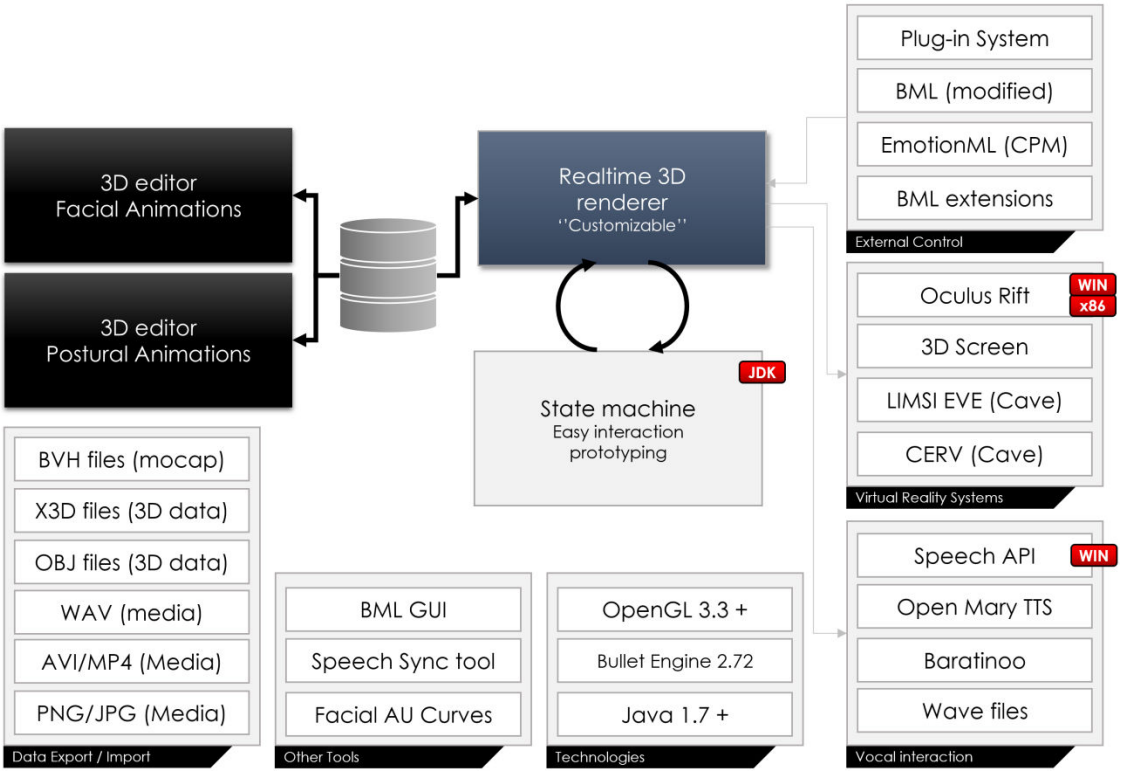

Figure 25. General Overview on MARC Architecture

2.3.3 SAIBA Framework

The multimodal information that can be obtained from the user must be integrated in the architecture of the embodied agents to improve their interaction performance. Most ECA systems, such as Greta, MARC and Virtual Human Toolkit, worked on finding a common real-time multimodal behavior generation framework that can independently generate communicative functions to be realized by these ECAs [89]. Accordingly, the SAIBA (Situation, Agent, Intention, Behavior and Animation) framework was created [90] [91].

SAIBA is a framework that can be integrated in an ECA system to generate natural multimodal behaviors. This framework takes into consideration how the agents should reason about what it has to do or say. It generates the natural multimodal communicative intentions and behaviors, which should be realized by the ECA, and represents it using normalized languages.

The SAIBA framework is composed of three separated levels of abstraction: 1) representing the planned communicative intentions (Intent Planner), 2) planning for a multimodal realization by selecting the communicative behaviors of planned intentions (Behavior Planner), and 3) the realization of the communicative behaviors (Behavior

Realizer) on a virtual agent (Figure 26). However, the separation of communicative intentions and the realization of its behaviors is supported in SAIBA through two interfaces using the Function Markup Language (FML) [92] [89] and the Behavior Markup Language (BML) [91] [90] respectively.

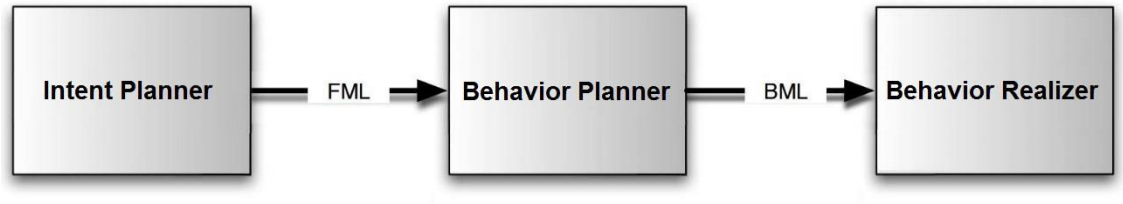

Figure 26. SAIBA framework for multimodal generation

The FML is used to represent the communicative intentions for the interface between the first two stages (Intent Planner and Behavior Planner) in order to plan for the communicative behaviors in the second stage. Likewise, the BML is used to represent the multimodal behaviors in the interface between the Behavior Planner stage and the Behavior Realizer stage.

2.3.3.1 Intention Planner: planning of communicative intents

All knowledge patterns defined in the cognitive architecture can be used by the Intent Planner module of SAIBA to determine the communicative intentions of the agent and without any reference to physical behaviors. The communicative intentions consist of performative actions, emotions and references to the context of the virtual environment. These intentions are coded using the FML script. This language still lacks a unified standard, although several systems following this framework are trying to propose their own version [89].

The properties of existing ECA systems helped in structuring the FML script to represent the selected communicative intents. The FML is created based on the Multimodal Utterance Representation Markup Language (MURML) used in MAX [93], and mainly on the FML-APML (FML-Affective Presentation Markup Language) [94] which was created for Greta. However, the Virtual Human Toolkit also uses an FML-like concept to generate the communicative intents of its agents.

Based on these contributions, an FML specification is proposed by [89] with several important components:

1- Contextual information and person characteristics: Includes detailed information about the semantics of the virtual environment and the characteristics of the user that is interacting with the system.
2- Communicative actions: Verbal and non-verbal communicative actions are chosen so that the agents can handle their specified roles.
3- Emotional and mental states: Suitable emotions are selected to increase the motivational effect of communicative intents, and the mental states such as remembering and planning are considered as cognitive processes.

Accordingly, the FML considers several important matters to define the communicative intentions:

1- Various contextual information
2- Classifying communicative actions into categories
3- Distinguishing planned intents from unconscious intents, like emotions
4- Dividing the communicative intents into small chunks with particular timing information in order to be processed separately
5- Defining the structure of the FML script in an XML-like syntax with rules to embed tags
6- Dealing with one or several ECAs relatively
7- Handling one or multiple roles to the agents

The proposed FML specification is considered preliminary and has many limitations, but it facilitates generating the communicative behaviors by transforming the FML chunks contents into a BML script (Figure 27).

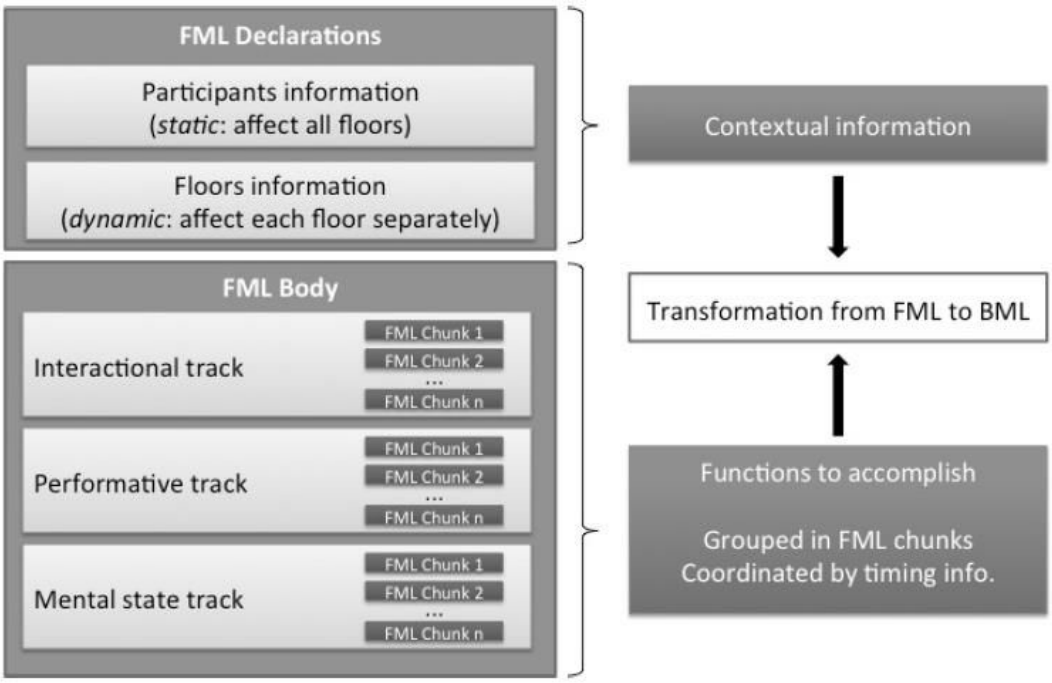

Figure 27. Proposed FML Specification

2.3.3.2 Behavior Planner: planning of communicative behaviors

The communicative intentions are parsed by the Behavior Planner module, where multimodal behaviors are automatically selected and coded using the internationally adopted BML language [95]. The properties of these behaviors and their timing information are described in this BML script. It is then forwarded to the Behavior Realizer to be realized by an ECA [90].

The communicative Behavior Markup Language (BML) is an XML based language that is used to represent and coordinate different types of verbal and non-verbal behaviors including speech, gestures, facial expressions, and body movement [91]. The Behavior Planner computes the start and the end timing information of each communicative behavior in order to be realized consequently. For instance, here is a simple BML script that could be generated by the Behavior Planner (Figure 28):

```
<bml>
   <face id="smile" start="2" end="5">
      <description level="1" type="agentbml"/>
   </face>
   <gesture id="greeting" start="2" end="5">
      <description level="1" type="agentbml"/>
   </gesture>
</bml>
```

Figure 28. A generated BML script

The Behavior Planner describes the multimodal behaviors using BML elements with necessary attributes (Table I). Various elements are commonly used to define the properties of the communicative behaviors that should draw the attention of the user interacting with the ECA [90]. The BML message containing the generated BML elements is then broadcasted to the Behavior Realizer which generates alternative animations.

Table I. The BML behavior elements

BML Element	Description
<head>	Movement of the head, like nodding and shaking
<torso>	Movement of the spine and shoulder
<face>	Movement of facial muscles to form certain expressions, like eyebrows and mouth movements
<gaze>	Coordinated movement of the eyes, neck and head direction, to indicate where the character is looking
<body>	Full body movement, like changing position and posture
<legs>	Movements of the body elements downward from the hip, like legs including knee, toes and ankle
<gesture>	Coordinated movement with arms and hands, including pointing and reaching
<speech>	Verbal behaviors, including the words to be spoken
<lips>	Controlling lips shapes including the visualization of phonemes

2.3.3.3 Behavior Realizer: realization of the planned behaviors

The Behavior Realizer in SAIBA framework is a realization engine that can realize all aspects of the multimodal behaviors which are scheduled by the Behavior Planner [90]. Such engines are BML-realizers that can compile BML scripts.

The behavior realizer uses a text-to-speech module to realize the speech of the verbal planned behaviors. In consequence, the animations of the movements of the ECA are directly generated upon receiving the BML messages from the Behavior Planner. The temporal constraints assigned to the planned behaviors play a significant role in

organizing its execution [90] [91]. The ECA platforms that are complaint with the SAIBA framework, such as Greta, MARC and the Virtual Human Toolkit, are considered as the Behavior Realizers.

3 MODEL

In this chapter, we present the architecture of our model. As introduced in the previous chapter, we extended the meta-model MASCARET. We aim at adding a BDI-like cognitive module, able to reason about the environment, which allows the agents to make decisions about their high-level intentions. An embodied agent can also have communicative intentions that must be transmitted to the user through natural communicative channels, such as voice, facial expressions, gestures etc. To achieve this goal, ECA systems based on the SAIBA framework are integrated to MASCARET.

3.1 Global Architecture

Figure 29 shows the overall architecture of our model. It is entirely based on MASCARET and it extends this meta-model by adding two more modules: (i) a virtual character integrator module (Figure 29 in green), which allows the integration of ECA system that are SAIBA compliant, and (ii) a BDI-based cognitive module (Figure 29 in blue), which determines the virtual character's intentions and desires according to its knowledge (beliefs).

We remind that one of our aims consists in introducing an Embodied Conversational Agent in the Virtual Learning Environment in order to allow the user to communicate naturally while interacting with the system. To achieve such a goal, we are not going to implement a brand new ECA since several virtual agent systems already exist and provide all communicative capabilities we need.

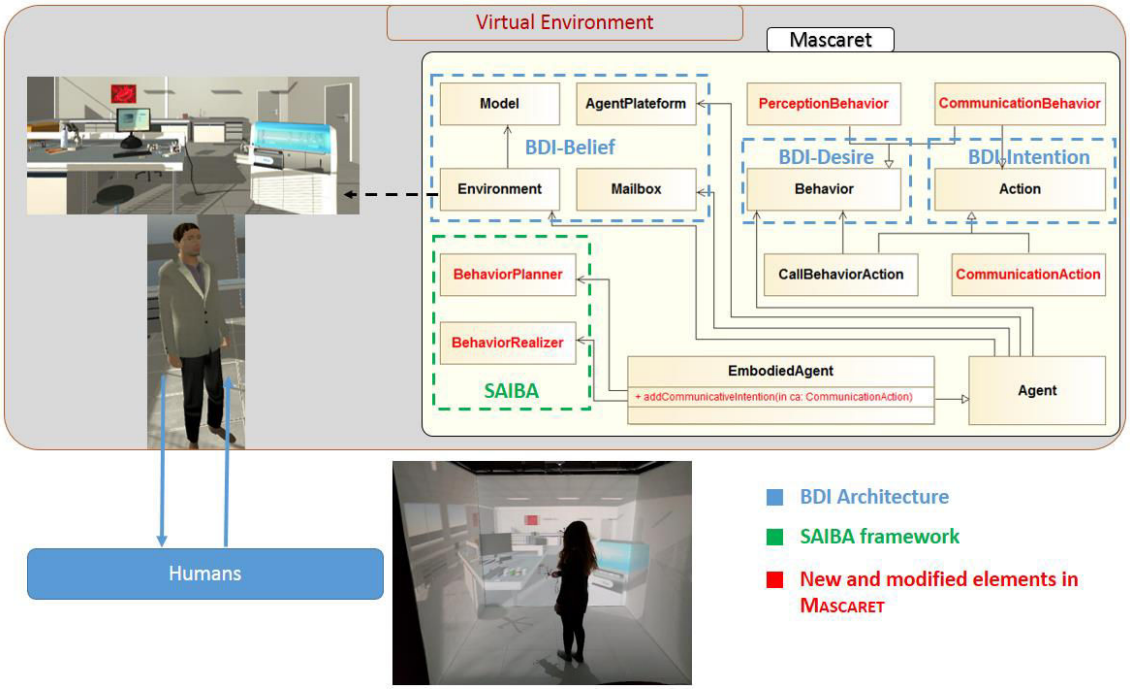

Figure 29. Global view of our model

What we want is to make it easy to integrate any ECA system that is compatible with the standard SAIBA framework. As explained in section 2.3.3, this framework divides the process to generate the virtual agent behavior into three levels of abstraction: from the selection of the agent communicative intentions, to the choice of the multimodal signals needed to transmit them and the signals realization on the virtual character graphical representation. These three levels, respectively, the Intent Planner, the Behavior Planner and the Behavior Realizer, communicate through standard languages: the FML, which codes the communicative intentions and the BML, which codes the multimodal signals [89] [91].

We have several reasons why we want to make it possible to integrate any SAIBA compliant ECA architecture in our model. Firstly, the systems built on a common standardized framework can be easily integrated. To generate the agent communicative behavior, all existing ECA systems based on SAIBA need to receive FML and/or BML messages according to the levels of abstraction they provide. For example, the Greta platform [76] implements both the Behavior Planner and the Behavior Realizer, which means that it can generate the agent behavior from FML and BML messages. MARC [87] and the Virtual Human Toolkit [80] provide a Behavior Realizer, which means that they can generate the agent behavior described in the BML messages they receive. Secondly, the existing ECA systems can be embedded with different communication

capabilities, like speaking and human-like body movements, so we prefer to make it possible to choose the ECA that would fit better for a given application. Thirdly, the ECA systems provide different graphical representation of virtual agents that can be displayed through diverse players and in diverse devices (such as CAVE, HMD, screen...). For instance, Greta platform provides several agents of different gender that can appear in an Ogre player or in a Unity3D application; while MARC provides many tools to create new characters using a custom graphic engine based on OpenGL and other open-source API. So, we want our model capable of integrating the appropriate ECA system according to the virtual reality device and the graphical agent representation we need for a given application. Finally, we hope to incite ECA researchers to easily use and test our model.

Existing ECA platforms which are SAIBA compliant, provide mainly an implementation for the Behavior Planner and/or the Behavior Realizer. For such a reason, to allow the integration of these platforms in MASCARET, we propose to implement two interfaces, one for the Behavior Planner (BEHAVIORPLANNER) and the other for the Behavior Realizer (BEHAVIORREALIZER) (Figure 29 in green). The details of this integration are presented in section 0.

Another main goal of this work is to formalize the intention of the agents. To achieve this goal, we implement a cognitive module within MASCARET inspired by BDI architecture that permits us to generate high-level intentions for the agents (Figure 29 in blue). **Beliefs** are considered as the knowledge base of the agents. It consists of references to domain concepts (like the structure and the properties of the defined entities of the considered domain), environment topology (such as the position of entities) and activities of all agents in the environment (including the user).

As described in chapter 2, the "**desires**" of an agent can be considered as its higher goal to reach. To attain this goal, the agent organizes lower level goals (**intentions**). An "**intention**" can modify the higher-level goal of an agent. We consider that this recursive design pattern can be implemented through behaviors (Figure 29, BDI-Desire) and actions (Figure 29, BDI-Intention) in MASCARET (BEHAVIOR and ACTION classes). This hierarchical and recursive design pattern between **desire** and **intention** permits to develop complex behaviors for the agents. We consider in this work that in MASCARET the desire will be represented by BEHAVIOR and intention by ACTION. The communicative intention (actions) are generated by those complex behaviors. For example, when the agent is embodied, communicative actions are planned to transmit

its communicative intentions. So, for embodied agents which need to communicate with the user, we propose to implement the Intent Planner of SAIBA.

We propose a specific type of action to represent communication between agents in general and communicative intention in particular (intentions). By representing this communicative action in MASCARET metamodel (COMMUNICATIONACTION class), it permits to use this action in higher behaviors which represent the desires of the agents (desires). We present how we implement these concepts in section 3.3.

The knowledge base of the agents (beliefs) is used to permit them to reason on their environment. We modify MASCARET to formalize the knowledge base of the agents based on the ontology of the organization and activity and the exchanged messages in the virtual environment (see Chapter 2). The detailed definition of the knowledge base is presented in section 3.2.

The knowledge base of the agents has to constantly be updated. Therefore, we developed two ways of updating the knowledge base of the agents: through perception and through communication. Embodied agents can perceive the states of entities while navigating in the virtual environment, or they can communicate with each other to exchange information and share their knowledge.

Implementation details of our architecture are provided in the following sections and the next chapter shows how this generic architecture can be instantiated to implement a pedagogical situation in a Virtual Learning Environment which involves a tutor agent and a learner.

3.2 Model of Knowledge

According to the BDI agent architecture (discussed in section 2.2.1), each agent holds its own “beliefs” including information about itself, other agents and the contents of its environment [96]. In BDI, a distinction between beliefs and knowledge is considered. In beliefs, some assumptions about accuracy and implicitness can be carried out by the agent on the gathered information (knowledge) [52].

Domain experts usually define the domain model of a real system. They can provide the most accurate representation of the system in order to simulate it in the virtual environment. In MASCARET meta-model, the ENVIRONMENT and the MODEL classes, which are held by the MASCARETAPPLICATION singleton, are used to hold this domain expert representation (Figure 30).

In this work, we explicitly add an association between the AGENT and the ENVIRONMENT classes to represent the knowledge of agents. We can notice that our system allows to have several agents referencing different instances of ENVIRONMENT and MODEL classes. This is to say that in our model agents can coexist in the same virtual environment and they can have different knowledge on it. This knowledge can be different from the model held by MASCARETAPPLICATION. This way of representing the knowledge can be seen as the beliefs of agents.

In the following sections, we discuss what the knowledge base of agents includes, and how it is fetched, exchanged and evolved.

3.2.1 Components of Knowledge

The knowledge base of the agents has the same structure of the ENVIRONMENT of MASCARETAPPLICATION, but it includes some additional components: organizations (agents and activities) and exchanged messages.

Hence, three knowledge components constitute the knowledge base of the AGENT: the structure of the environment, the agents and organizations in the environment, and the messages in the mailbox. We modify Mascaret by creating associations in the meta-model (in red in Figure 30) to represent these components.

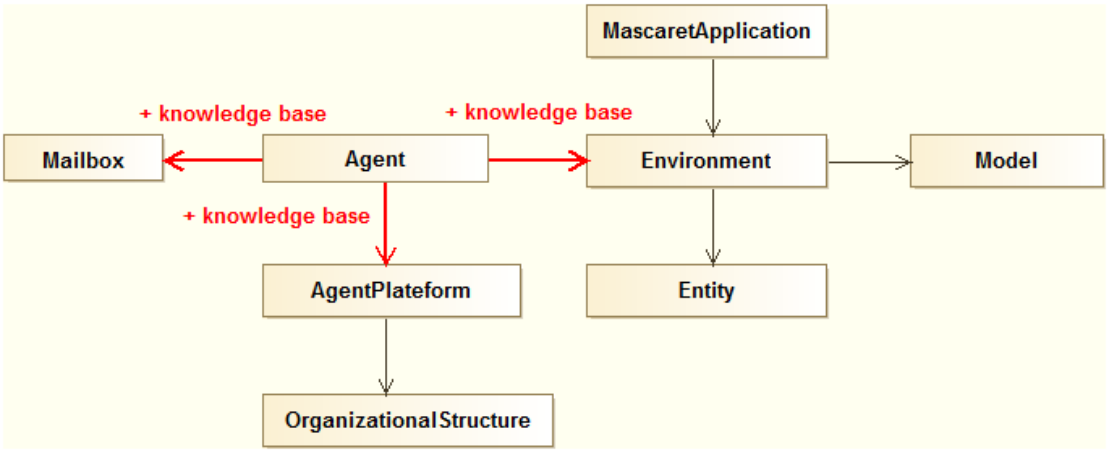

Figure 30. The knowledge base of the Agent

3.2.1.1 Structure of the Environment

The knowledge base of the agent is represented by the same structure used to describe the real system through the MASCARETAPPLICATION and its ENVIRONMENT and MODEL. Therefore, the agent knowledge base refers to an ENVIRONMENT and its MODEL (Figure 31). The detailed content of those classes has been described in Chapter 2. We just

remind here some essential components held by the ENVIRONMENT and MODEL from MASCARET to emphasize the content of agent's knowledge base using this model.

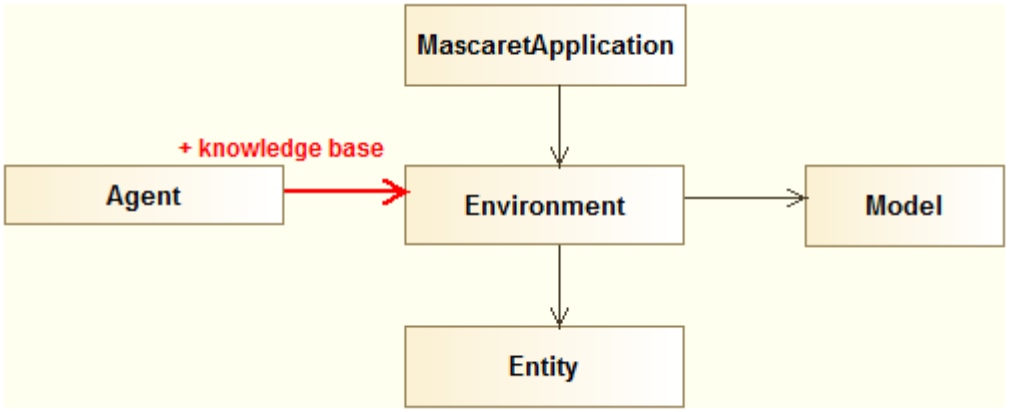

Figure 31. The structure of Agent's Environment

In summary, an ENVIRONMENT (and then the knowledge base of the agent) is composed of entities and their properties. It also contains information on the geometric representation, position, animation and topology of entities. It refers to a MODEL that describes the structure of the ENVIRONMENT based on the classes, properties and relations. The documentation on the instances of those concepts is also maintained in the model and can be used as a knowledge base for the agents. For example, when an agent is asked for the definition of an object, it can introspect its knowledge base and get this description.

Each class in the MODEL can have a state machine to describe the autonomous asynchronous reactive behaviors of the entities of this class. All the concepts in this state machine can also be documented and used as a knowledge base for the agent.

3.2.1.2 Agents and Organizations

Agents in MASCARET are aware of further agents and organizations found in their environments (through AGENTPLATFORM, see Chapter 2). In our model, we also use this as a knowledge base for the agents. Each agent has a reference to an AGENTPLATFORM that enriches its knowledge base with a set of organizations (ORGANIZATIONALENTITIES: instances of ORGANIZATIONALSTRUCTURE) and agents in its environment. Here again, we do not add concepts in this structure, we just remember some important elements that are used as knowledge by agents.

The ORGANIZATIONALSTRUCTURE provides the knowledge base of the agents with the structure of ORGANIZATIONALENTITY through PROCEDURES, RESOURCES and ROLES

(Figure 32). For example, when the user asks about the next action in the procedure, the AGENT checks the organization in the AGENTPLATFORM in its knowledge base and gets the name of the next action from the PROCEDURE.

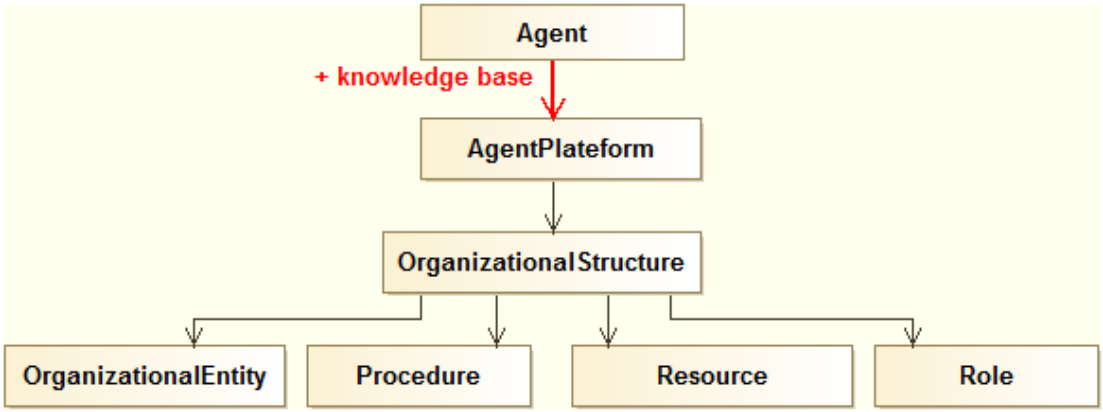

Figure 32. The platform of the Agent

3.2.1.3 Mailbox

Agents communicate with each other by exchanging messages. They constantly archive the exchanged messages through instances of the MAILBOX class in MASCARET. In our model, we also use this principle as a knowledge base for the agent.

The MAILBOX class includes different types of messages: *ReceivedMessages*, *CheckedMessages* and *SentMessages* (Figure 33). Initially, messages received by an agent are stored in the *ReceivedMessages* list. When the agent checks the contents of a received message, the message is directly moved to the *CheckedMessages* list. While the messages that the agent sends are stored in the list of *SentMessages*. This process is made by default in MASCARET.

In our model, an agent considers the different types of messages stored in its MAILBOX as a knowledge base. For example, when an agent explains the next action to be done by the user, it sends a message and stores it in the *SentMessages*. Then, if it receives a new message from the user asking for the goal of this action, it uses the sent message to get the information about what is the requested action.

The way the agents interpret contents of messages will be explained in section 3.3.2.1.

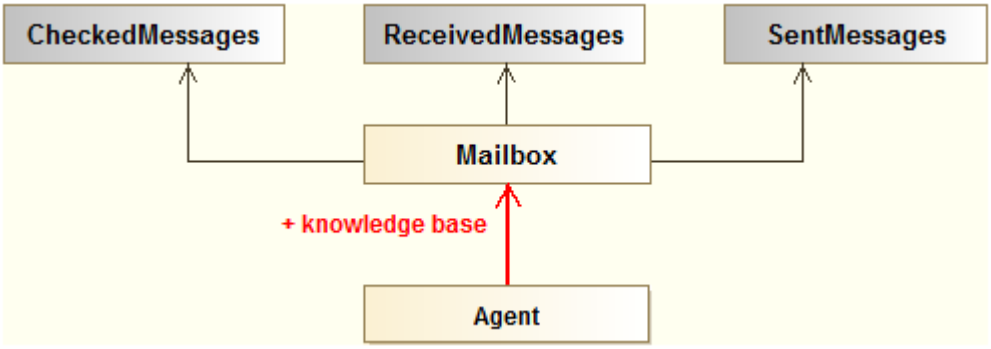

Figure 33. The mailbox of the Agent

3.2.2 Initial instantiation of knowledge base

All agents in MASCARET can share the same instance of the ENVIRONMENT as a knowledge base and then refer to the ENVIRONMENT referenced by the MASCARETAPPLICATION. This permits to save time and memory during the simulation. It also permits to solve consistency problem about the actual state of the ENVIRONMENT and the knowledge base of the agents. Nevertheless, it is not realistic for agents to gain the full knowledge of the system while interacting with the user. Thus, we propose a model that permits each agent to hold its own instance of the ENVIRONMENT and the MODEL classes within its own knowledge base. This knowledge base has to be constantly updated whenever changes in its environment are perceived.

Consequently, agents in our system might carry a total or partial notion of the real model held by the MASCARETAPPLICATION. The knowledge base of each agent can then be seen as its own beliefs about the environment.

Three knowledge design patterns can be instantiated for the knowledge base of the agents:

1. Agent that shares the ENVIRONMENT instance referenced by the MASCARETAPPLICATION and structured in the MODEL instance. This agent has a complete and exact knowledge on the simulated ENVIRONMENT. We can notice in Figure 34 how *Agent1* has in its knowledge base the same instances of the ENVIRONMENT (*Environment1*) and the MODEL (*Model1*) referenced by the MASCARETAPPLICATION singleton (*MascaretApplication1*).

   As an illustration, when an instance of an entity is created in *Environment1* using the structure specified in *Model1*, the knowledge base of *Agent1* is automatically consistent.

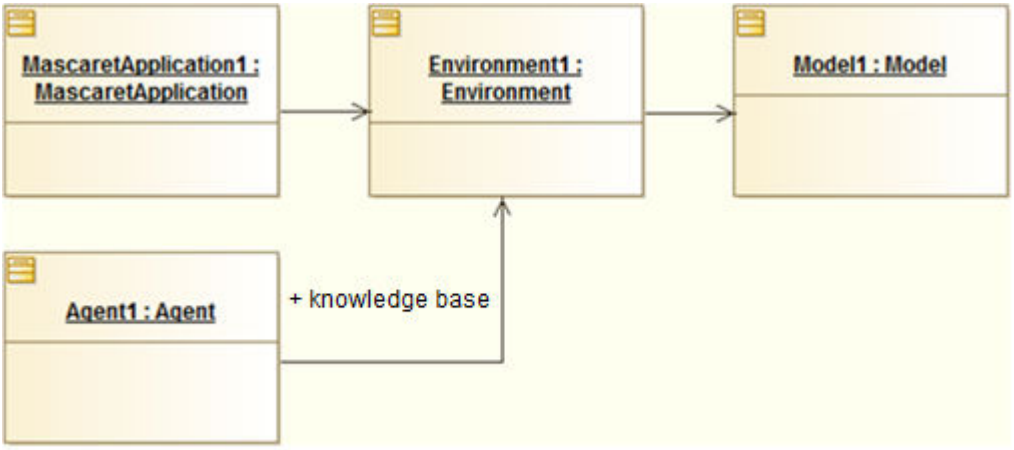

Figure 34. Agent with complete knowledge

2. Agent referencing its own instance of the ENVIRONMENT (*Environment2*) but shares the same structure in the MODEL instance (*Model1*) of the *MascaretApplication1* (Figure 35). When a class of a certain entity is defined in *Model1*, various instances with the same structure of this entity can be established in *Environment1* and *Environment2*.

   Furthermore, *Agent1* is unaware of anything found outside its environment (*Environment2*), but surely abides by the structures found in *Model1*. For instance, when we specify the structure of a "BOX" entity in *Model1* with the "Color" and "Size" attributes, instances of this entity with different "Color" and "Size" values can be set in *Environment1* and *Environment2*. However, *Agent1* will only hold the knowledge of "BOX" instances found in *Environment2*.

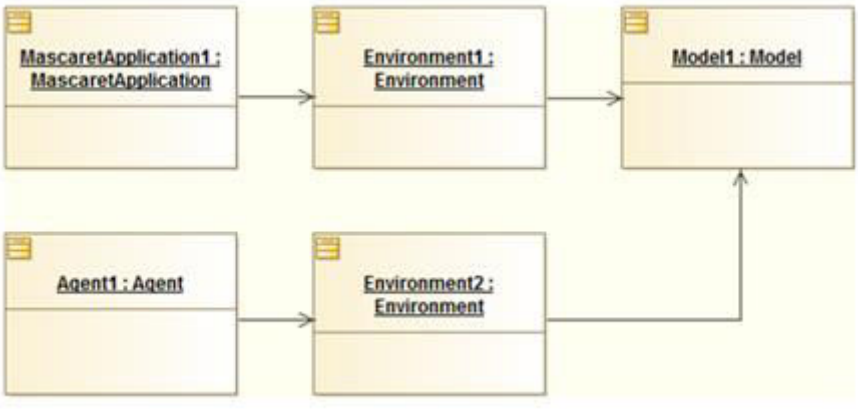

Figure 35. Agent with individualized Environment of a common Model

3. Agent carrying the knowledge about its own environment (*Environment2*) based on a different MODEL (*Model2*) (Figure 36). In this case, same states and results

could be attained on entities found in *Environment1* and *Environment2*, but distinct activities defined in *Model1* and *Model2* are considered.

For example, suppose that instances of the tool shown in Figure 37 are set in *Environment1* and in *Environment2*, and the user had to adjust its "HOUR" value. The relevant functions implemented in *Model1* and *Model2* could ask for different requirements. The function of *Model1* could require the "CLOCK" button to be pressed in parallel with the "HOUR" button in order to increment its value, while that of *Model2* could recommend from *Agent1* that the "CLOCK" button has to be pressed and released before the "HOUR" button is pressed.

This principle permits not only to have more realistic behaviors as agents may not have a complete knowledge on the environments, but it permits also to let several models of the same system (from several experts) co-exists in the same environment.

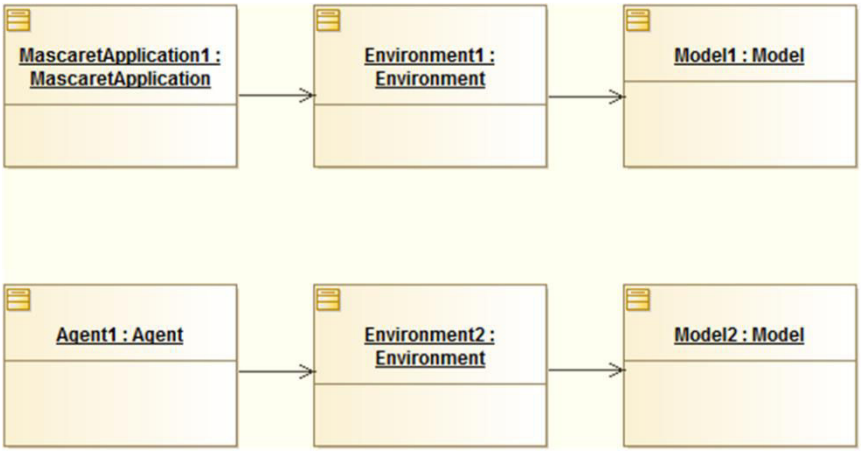

Figure 36. Agent with particular Environment and Model

Figure 37. Environment tool

3.3 Behaviors and Actions

As described in chapter 2, the "desire" of an agent can be considered as its higher goal to reach. To attain this goal, the agent organizes lower level goals ("intention"). An "intention" can modify the higher-level goal (Figure 40). We consider that this recursive design pattern can be implemented through behaviors (BDI-Desire) and actions (BDI-Intention) in MASCARET (BEHAVIOR and ACTION classes). In the next section, we propose a generic behavior for an agent to actualize its knowledge through its perception when navigating in the virtual environment.

In other words, in MASCARET, the behaviors of the agent generate the execution of the actions (Intention). In MASCARET, some generic behaviors are implemented. The PROCEDURALBEHAVIOR is one example. This permits to execute activities (procedures) that organize *a priori* actions. As we focus on communication between the user and the agents, those actions can be communication actions (intentions). In the following of this chapter, we present a formalization of communication action that can be inserted in an activity.

However, MASCARET allows also to implement new behaviors. We then propose a PERCEPTIONBEHAVIOR in order for an agent to update his knowledge base (see section 3.3.1) and a COMMUNICATIONBEHAVIOR that permits to an agent to understand messages from other agents (or the user) and generate answers (execute a COMMUNICATIONACTION) (see section 3.3.2).

To be clear, according to the SAIBA framework, in our model we consider that the set of behaviors that the agent is running represents its Intent Planner. Those behaviors will generate communicative intentions (scripted in an activity or as the result of a deliberative behavior). This communicative intention will then be executed by SAIBA modules.

3.3.1 Perception behavior

As seen in section 3.2.2, agents can have their own beliefs on the environment. As this environment is an open environment (human user may manipulate entities), we propose a generic behavior for the agents to update their knowledge through perception when navigating.

In MASCARET, the embodied agents are defined using the EMBODIEDAGENT class that inherits the AGENT class and has a body (Figure 38). For this purpose, we add a field of

view (angle and distance) in the EMBODIEDAGENT class of MASCARET (Figure 38). We could propose other way to perceive the environment, like ray casting for example, but this is not the purpose of our work. Using its perception, an agent can detect new entities and add them in its knowledge base or update properties of already known entities. This behavior is useful when the agent knowledge base does not refer the same instance of ENVIRONMENT as MASCARETAPPLICATION as seen in section 3.2.2. We consider that as soon as an agent perceive an entity, it has access to all its properties. The following example shows how we implement the evolution of the knowledge through perception.

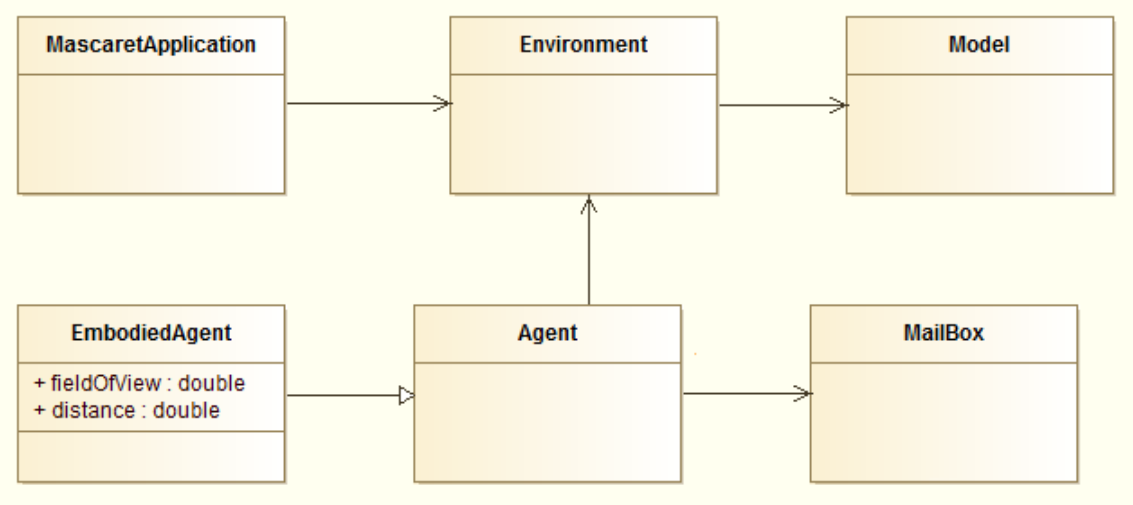

Figure 38. Embodied agent with detection properties

When we define two agents (Astrid and Bruce) in the virtual environment (shown in Figure 39-a) with the knowledge about existing entities (Box1 and Box2), the initial knowledge of Astrid and Bruce includes the states (Box1[closed] and Box2[closed]) and the initial values of the properties of these boxes, such as their color (Box1-color=”Brown” and Box2-color=”Blue”). Let us suppose that a realized behavior changed the state of Box1 to “opened” and its color property to “Red” (Figure 39-b). Since Box1 is in the field of view of Bruce, he directly detects this variation and automatically update his knowledge base with the new state and color value of Box1. While Astrid does not notice this change since Box1 is not in her field of view. However, when Astrid navigates in the environment and sees Box1 in her field of view (Figure 39-c), she updates her knowledge base with the detected changes in the state and the properties of Box1.

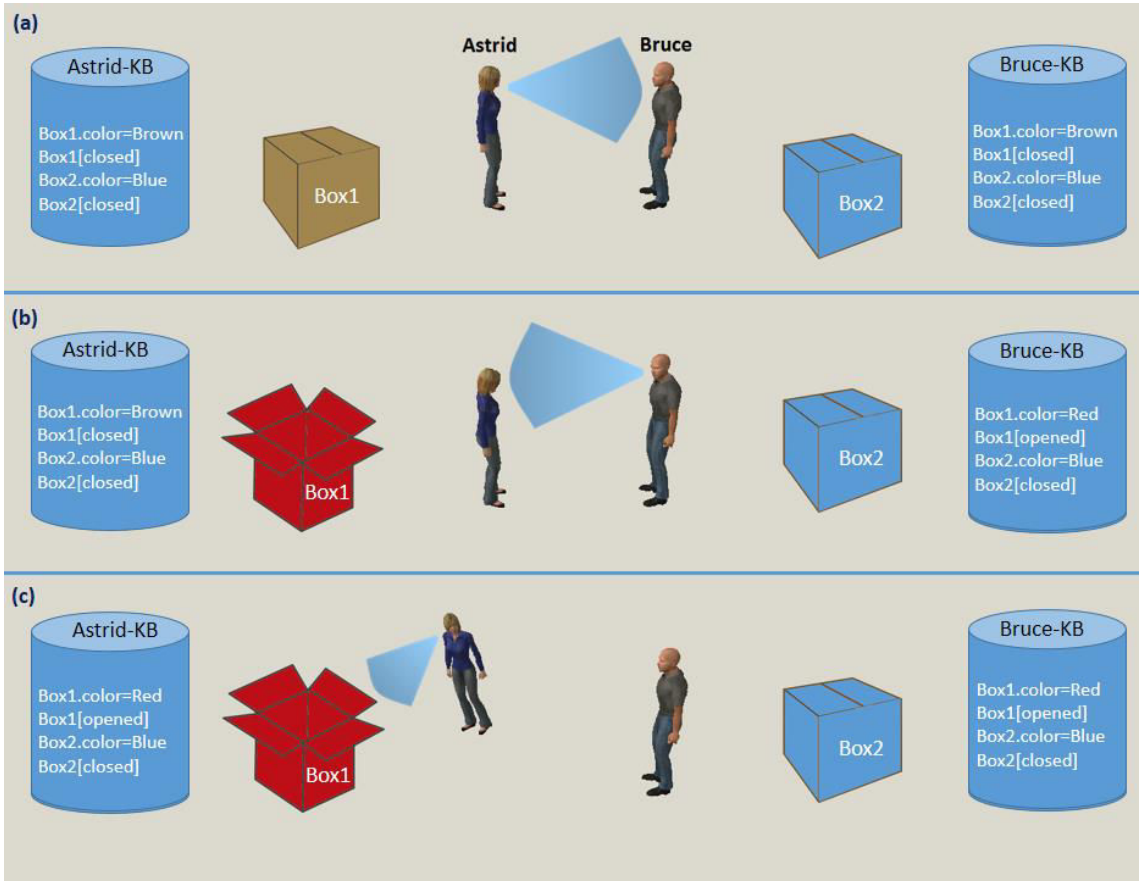

Figure 39. The evolution of agent's knowledge through perception

3.3.2 Communication Action

As seen before, we use the design patterns "Behavior" and "Action" to represent the concepts of "Desire" and "Intention" of the agents in MASCARET. In our work, we mainly focus on the communication between agents.

MASCARET is founded on UML metamodel, and proposes several types of actions (classes that inherit from the ACTION class) like CALLOPERATIONACTION, which is an elementary operation that the agent calls to execute an operation on an object in the environment, and CALLBEHAVIORACTION, which triggers another behavior with a set of necessary actions. These actions can be used in behaviors, like activity. MASCARET uses the ACTIVITY class to describe the procedures (domain model or pedagogical scenario for example).

We propose here a new type of ACTION to represent the communication action (COMMUNICATIONACTION class), or in terms of BDI and SAIBA, the communicative intention (Figure 40). This COMMUNICATIONACTION can be used in behaviors.

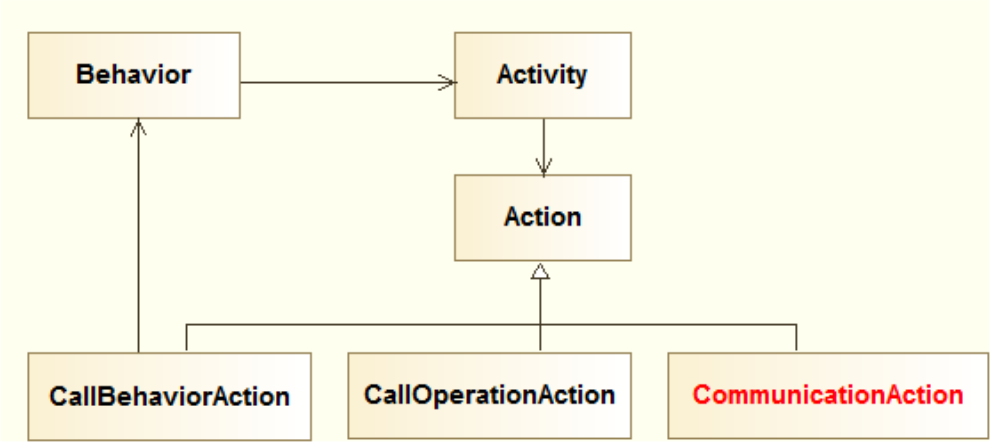

Figure 40. The actions for the activities of agents

For example, when executing an activity where the agent needs to show an object in the environment to the user, it uses the CALLBEHAVIORACTION "Explain [ObjectName]" to trigger a new behavior (Figure 41). The activity of the "Explain" behavior uses the CALLOPERATIONACTION "Highlight (ObjectName)" to highlight the specified object and the COMMUNICATIONACTION to inform the user with the description of the activity.

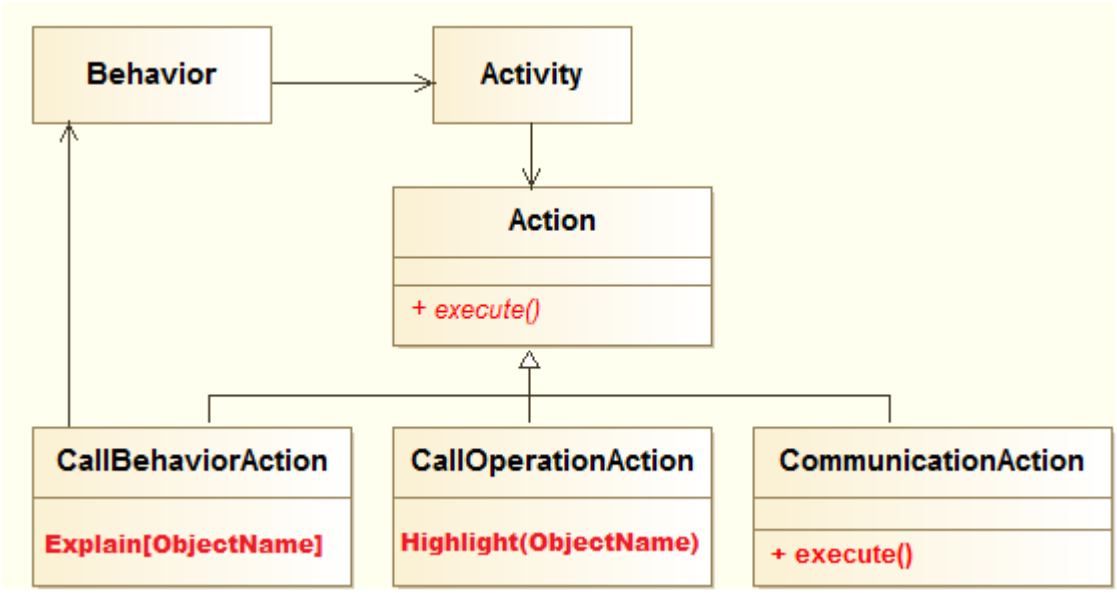

Figure 41. The actions of a complex behavior

At the AGENT level, we instantiate the COMMUNICATIONACTION class to perceive the sense of communication between agents. While at the EMBODIEDAGENT level we use the COMMUNICATIONACTION to naturally communicate with the user through the behaviors of the connected ECAs (emit/perceive voices, gestures, natural language, lip sync…).

When a behavior triggers a communication action (execute() on a COMMUNICATIONACTION), the agent running the behavior first sends a FIPA-ACL message according to the content and the context of the communication action. If the agent is an embodied agent (remember that EMBODIEDAGENT is a subclass of AGENT), it adds this communication action as a new communicative intention (using the *addCommunicativeIntention* operation in EMBODIEDAGENT) to be managed by our SAIBA integration.

The first level (AGENT level) is presented in this section, while the latter (EMBODIEDAGENT level) is presented in section 3.4.2.3.

3.3.2.1 Agent communication protocol and content language

Among the existing agent communication languages, MASCARET uses the FIPA-ACL as a communication protocol between its autonomous agents, and considers the FIPA-SL[7] (Semantic Language) as the message content language.

The fundamental properties of the FIPA-ACL protocol are shown in Table II.

Table II. Parameters used in FIPA-ACL messages

Parameter	Category of Parameters
performative	message label; type of communicative acts
Sender	message sender
Receivers	message receivers
Content	contents of the message
language	adopted language
encoding	encoding message
ontology	adopted ontology
protocol	message protocol
conversation-id	conversation unique identity

[7] http://www.fipa.org/specs/fipa00008/SC00008I.html (accessed November, 2017)

reply-with	expression used by replying agent to identify the message
in-reply-to	date/time label to indicate when an answer must be received
reply-by	reference to a previous action, where the message is an answer

The *performative* property has a fundamental role in building FIPA-ACL messages. It specifies the objective of the FIPA-ACL message and the speech act that an agent wants to deliver to another agent. Appendix 1 shows the twenty-three FIPA-ACL performative functions and their corresponding meanings. These functions propose a wide variety of request and response messages that the agents can exchange. For example, in our model, when an agent seeks for the value of an attribute of an object in the virtual environment, it must use the "QUERY-REF" performative function since it denotes making a request (see example in section 3.3.2.1).

In our model, we also manage the sender, the receivers and the message content of the FIPA-ACL messages. The effective content of the message can be formalized by a content language. We choose to use FIPA-SL and we explain it in the next section. The other properties of FIPA-ACL messages (conversation, in-reply-to,…) are out of our scope in this work.

The ACLMESSAGE class in MASCARET (Figure 42) allows to code messages according to the typical structure of the FIPA-ACL messages. The agent uses instances of the ACLMESSAGE class to build and parse exchanged messages with other agents.

The content property of a FIPA-ACL message is defined using the FIPA-SL (content language specification). In our model, we enriched the FIPA-SL parser in MASCARET, to parse the contents of the FIPA-ACL messages using the standard parsing rules defined by FIPA. The complete grammar (FIPASL.g4 in ANTLR) is presented in Appendix 2.

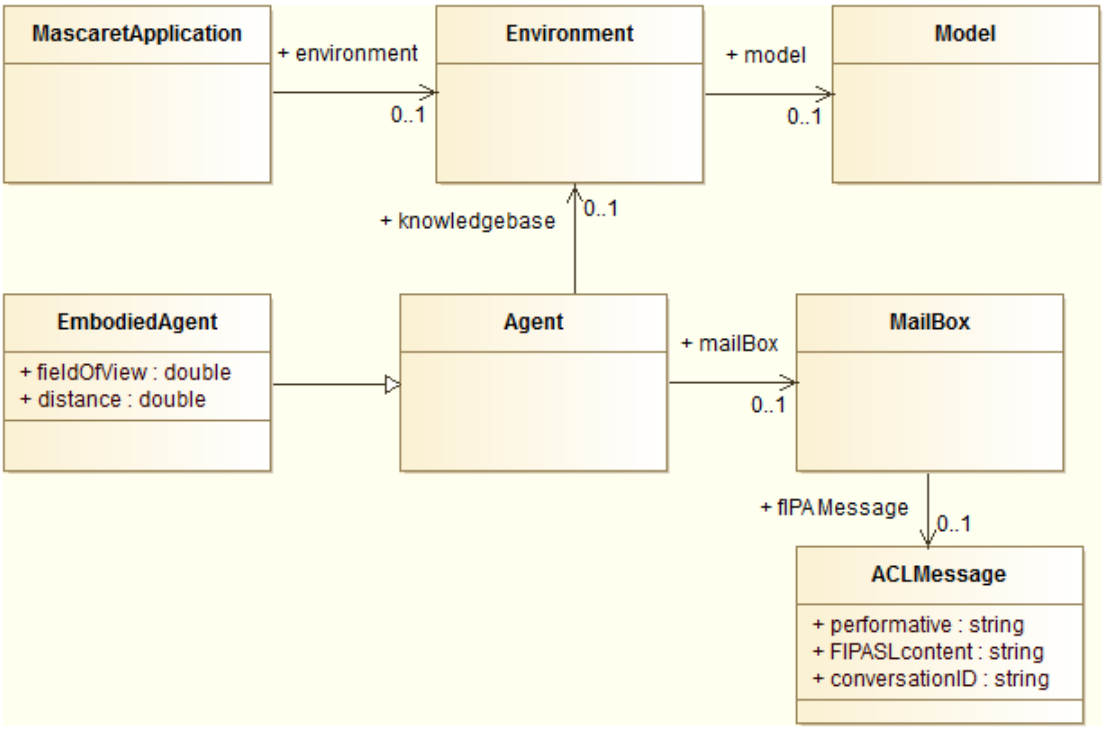

Figure 42. Agents in MASCARET communicate using the ACLMessage class

In this section, we just take one typical example of message content to explain how it works in our model. Among the referential operators defined in the FIPA-SL, the "iota" operator allows asking for a term (property) inside the knowledge base of an agent. The value of a property is stored in MASCARET in a "slot". Therefore, when an agent receives a FIPA-ACL message and detects the "iota" operator in its content, it recognizes that it should get the "slot" value of the specified entity. In the reply message, the "INFORM" performative function should be used to transmit the "slot" value.

For example, when "Agent1" in MASCARET receives a FIPA-ACL message from "Agent2" with the "QUERY-REF" performative asking for the position of "Agent3" in the environment, "Agent1" checks its knowledge base for the necessary information and replies with another FIPA-ACL message to "Agent2" using the "INFORM" performative function.

The FIPA-ACL request message received by "Agent1" would be:

```
(query-ref
 :sender (agent-identifier :name Agent2)
 :receiver (set (agent-identifier :name Agent1))
```

```
:content ((iota ?position (slot ?position ?Agent3)))
:language FIPA-SL0
:ontology Agent-Management
:protocol fipa-request
)
```

When "Agent1" succeeds to obtain the coordinates of the position of "Agent3" from its knowledge base (value of the slot "position" of entity "Agent3"), the reply message has this form:

```
(inform
:sender (agent-identifier :name Agent1)
:receiver (set (agent-identifier :name Agent2))
:content ((= (iota ? position (slot ?position ?Agent3)) 120,50,240))
:language FIPA-SL0
:ontology Agent-Management
:protocol fipa-request
)
```

3.3.2.2 Communication Action class

As we introduced in this section (3.3.2), we extended the ACTION class in MASCARET to develop the COMMUNICATIONACTION class in order to represent the communication between agents and by extension (in EMBODIEDAGENT class) the communicative intentions of the agents.

In the COMMUNICATIONACTION class, we defined properties (performative, receivers, resources, FIPA-SL content, and natural content) to specify the characteristics of the exchanged communication actions of the agents (explained in the previous section). Here is a brief description about these properties:

- Performative: the function that specifies the style of the action the agent has to realize.
- Receivers: the agents that should receive the communication action.

- Resources: the objects of the environment involved in the communication action.
- FipaSLContent: the content property of the FIPA-ACL message using the formal content language FIPA-SL (as explained earlier). This content can be automatically interpreted (and generated) by agents.
- NaturalContent: the natural language text that the agent can use to inform the user. This text cannot be interpreted by the agent. It will be only treated by the SAIBA integration. It will cause nothing in the environment.

As the COMMUNICATIONACTION inherits from ACTION, it can be used in BEHAVIORS in MASCARET. For example, it can be used as an ACTION in an activity.

UML activity diagrams are used in MASCARET to provide scenarios and procedures through a sequence of actions. Within these activity diagrams, we can assign the actions of the agents. In this work, we use Modelio as UML Modeler. Modelio permits to create UML profiles by the addition of stereotypes. Hence, to create the communication action of the agents in the UML model, we create an instance of the ACTION class and append a "**CommunicationAction**" stereotype to it (Figure 43). To specify the communicative intents of the agents, we defined in this "**CommunicationAction**" the same properties of the COMMUNICATIONACTION class in MASCARET (Performative, Receivers, Resources, FIPA and Natural).

The "**FIPA**" property is set to communicate with the agents and request actions from them. For example, when we require an agent to inform the user about the next requested action in the domain procedure, we set the following FIPA-SL content in the "FIPA" property: "*((action AgentName (NEXT)))*" (Figure 43). The agent can parse this message, gets the "Description" of the desired action, and transmits it to the user (agent playing the role "utilisateur" in the activity). In section 0, we will present how this message will be automatically generated in natural language.

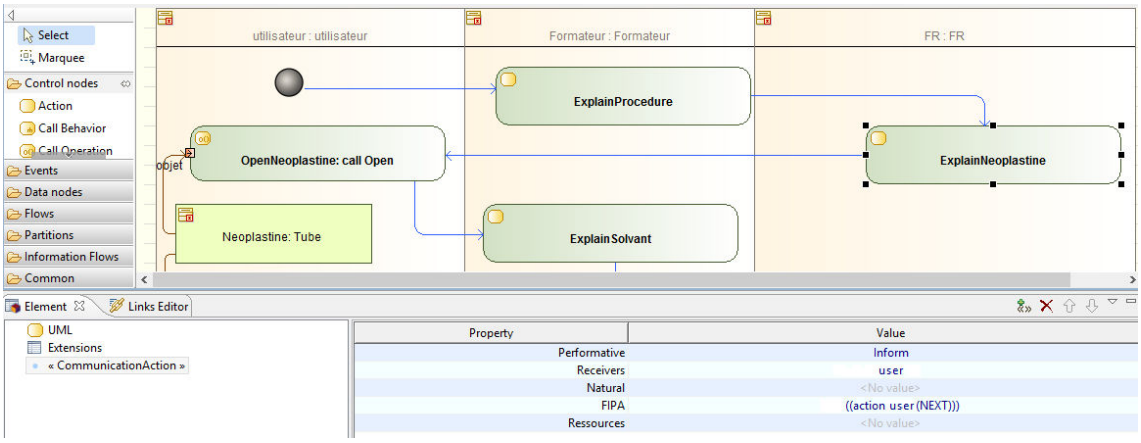

Figure 43. Instantiating the CommunicationAction in the UML profile

To directly inform the user with a specified natural text, we set the "**Natural**" property in the "**CommunicationAction"** in the activity diagram. If an embodied agent is realizing the communication action, it takes the value of the "**Natural**" property, and vocally tells it to the user through an ECA. However, to set a particular sentence, the person working on this profile can easily change the value of this property (Figure 44).

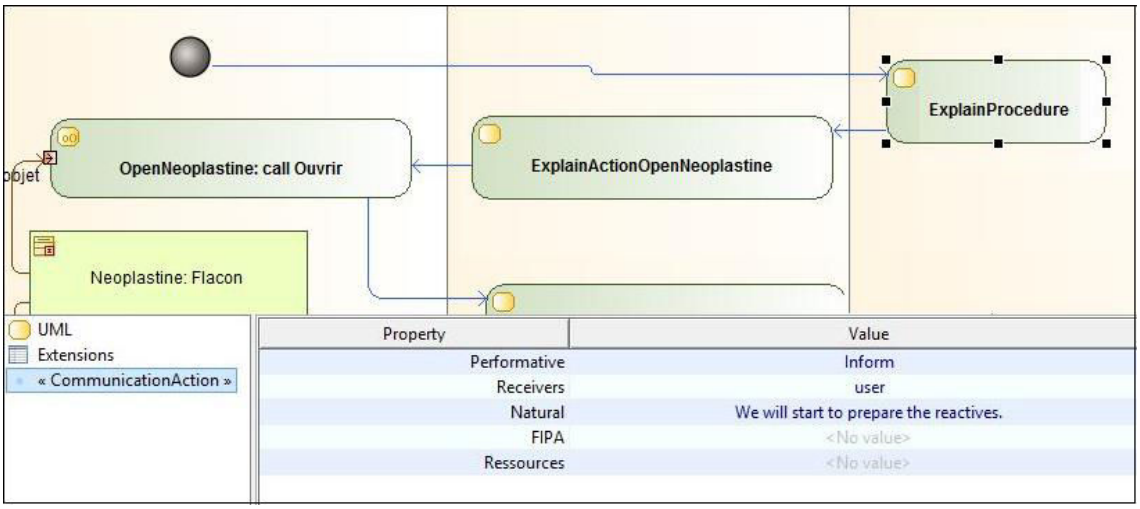

Figure 44. Setting a natural language text in the communication action

3.3.3 Communication Behavior

We propose a generic COMMUNICATIONBEHAVIOR to automatically manage the communication between agents. This behavior inherits from the AGENTBEHAVIOR class like other behaviors (desires) of the agents (Figure 45) such as the PROCEDURALBEHAVIOR (already proposed in MASCARET) that is defined to realize a scheduled activity.

This generic behavior is defined according to the formalization of the communication protocol (FIPA-ACL and performative) and the communication content (FIPA-SL syntax) proposed in the previous section.

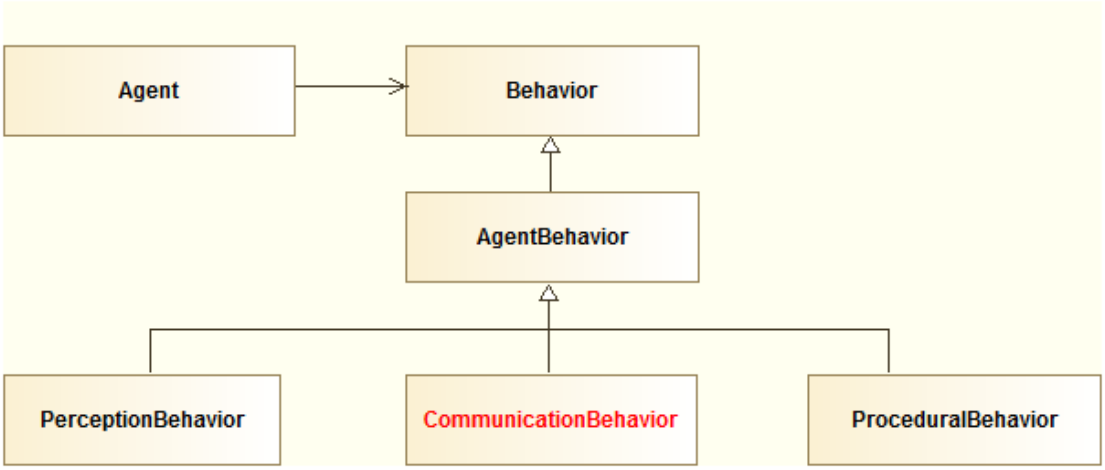

Figure 45. The behaviors of the agents

In this behavior, we didn't propose to manage all the 22 performatives of FIPA, as we focus on communication (transfer of knowledge between agents), we propose a generic behavior, called "action", to automatically manage three performatives: REQUEST, INFORM and QUERY-REF (Algorithm 1).

Algorithm 1: action()

```
1- //get next message
2- ACLMessage FIPAmsg = performer.receive()
3- if (FIPAmsg!= null)
4-      if (FIPAmsg.performative == REQUEST)
5-           manageRequest (FIPAmsg)
6-      if (FIPAmsg.performative == INFORM)
7-           manageInform (FIPAmsg)
8-      if (FIPAmsg.performative == QUERY-REF)
9-           manageQueryRef (FIPAmsg)
```

When the agent receives a FIPA-ACL message, it is added to the queue of the *ReceivedMessages* list in its MAILBOX. The agent gets the next message from this list and starts parsing it by checking its performative function (as seen in Algorithm 1).

The "REQUEST" performative is already used in MASCARET to synchronize the actions between agents, we just modified it by taking into account our proposition on FIPA-ACL and FIPA-SL. We do not present it here, but we will focus on INFORM and QUERY-REF for knowledge transfer.

When an agent needs to ask another agent about an object in the virtual environment, it builds a FIPA-ACL message using the "QUERY-REF" performative and the FIPA-SL expression that includes necessary operators. To form the FIPA-SL expression of the message, the agent defines a term using the required operators. In our model, we implement "Algorithm 2" in the COMMUNICATIONBEHAVIOR class to deal with exchanged FIPA-ACL messages between agents that use the "QUERY-REF" performative.

Algorithm 2: manageQueryRef

```
1- FIPASLParserResult result = parseFipaSL(FIPAmsg.content)
2- if (result.isIota){
3-    iota = result.iota
4-    if (iota.predicate == "slot"){
5-       //Manage Slot
6-    } else if (iota.predicate == "postcondition"){
7-       //Manage Postcondition
8- }
```

The FIPA-SL expression has 3 operators: *iota*, *any* or *all*. The "*iota*" is the reference operator that can be used to find a specific object that satisfies a defined property. While "*any*" indicates finding any object that satisfies this property. However, to find all objects that satisfies this property, the "all" referential operator can be used.

In our work, we are only using the "*iota*" operator to ask the agents about the properties of specific actions and entities (predicates), such as the name of an entity, the description of an action, the post-condition of an action, etc… For example, when the

agent needs to ask about any predicate of an entity, the following FIPA-SL expression is used:

((iota ?entity (predicate ?entity)))

When an agent receives a FIPA-ACL message and parses its FIPA-SL content, it realizes the required predicate and reply with the data in its knowledge base. When the "*iota*" operator is detected, the parser gets the value of the desired property and selects a behavior according to the value of the predicate like informing the requesting agent with this value using the following expression:

((= (iota ? entity (predicate ? entity)) value))

All properties of an entity are stored in slots. We are using the "slot" predicate to inquire about the parameters, the description and positions of entities, and the "postcondition" predicate to inquire about the goal of an action (Algorithm 2).

To manage inquiring about a slot of an entity, the agent has first to check if the slot exists in the specified entity (Algorithm 3). If the slot is recognized, the agent uses the "inform" parameter with the slot value to prepare the attributes of the COMMUNICATIONACTION (Performative and FipaSLContent) in order to reply to the inquiring agent and inform him with the value of the desired slot. The user could ask about the "position" of the entity (Where is *entity*?) in order to manipulate it, or she/he might ask about any property for this entity (What is the *property* of *entity*?), like "description", "color", "width", "height", etc… If the slot is not recognized, the agent uses the "NOT_UNDERSTOOD" performative and a natural text statement (NaturalContent) in the COMMUNICATIONACTION to inform to the requesting agent that the inquiry is not understood (Algorithm 3). Therefore, we show in the next section (3.3.4) how we are generically considering the inquiries of the user.

Algorithm 3: Manage Slot

```
1- object = findObjectInProcedure(iota.object)
2- value = getSlot(iota.parameter) //property of the object
3- if (value){ //slot value
4-    perf = INFORM   //performative
5-    naturalContent = value
6-    //FIPA-SL to give the slot value of the object
```

```
7-  FipaSLContent = "((= (iota ?perf (slot ?perf ?object))
                                naturalContent))"
8- } else {
9-  perf = NOT_UNDERSTOOD      //performative
10-  naturalContent = "I didn't understand. Can you repeat?"
11-  FipaSLContent = ""
12- }
```

The user might also ask about the goals of required actions. We use the "postcondition" predicate in the *iota* operator to determine such inquiry (Algorithm 4). When the "postcondition" predicate is recognized, the agent reviews the list of *SentMessages* in its MAILBOX to reply accordingly. The agent proceeds in parsing the inquiry if the previously sent message was to inform about a certain matter, otherwise, it replies with a "Not Understood" message. If this matter was to inform about an action in the procedure, the agent checks for the next action to be done by the agent that is asking (maybe the user) and gets its goal from the "Postcondition" stereotype that is set in the procedural scenario. The agent uses this attribute to prepare a COMMUNICATIONACTION to inform the inquiring agent about this property (Algorithm 4).

The agent also checks if the user is consecutively asking about the postcondition of an action (Algorithm 4). After reviewing the previously sent message, the agent could detect that it was for replying for an inquiry of an agent about a slot (using *iota*). To reply with a higher-level purpose for applying the action, the agent makes sure that the predicate of the previous message was the "postcondition" of the action. It can then inform the agent that is asking about the goal of the activity. However, if a different predicate was determined in the previous message, the agent replies with a "Not Understood" message. For example, it is illogical to ask "Why?" after asking for the name of an entity (Algorithm 4).

Algorithm 4: Manage Postcondition

```
1- msgCount = Agent.Mailbox.MessagesSent.Count
2- if (msgCount > 0) {
3-  lastMessage = Agent.Mailbox.MessagesSent [msgCount - 1]
```

```
4-  if (lastMessage.Performative == "INFORM") {
5-    contentLastMessage = lastMessage.Content;
6-    resultLastMessage = parseFipaSL(contentLastMessage);
7-    if (resultLastMessage.isAction) {
8-      actionToDo = findNextActionToDo()
9-      naturalContent = actionToDo.goal
10-      perf = INFORM
11-      FipaSLContent = "((iota ?actionToDo.name (postcondition
12-                                  ?actionToDo.name ?iota.object)))"
13-    } else if (resultLastMessage.isIota) {
14-      iotaLastMessage = resultLastMessage.iota
15-      if (iotaLastMessage.predicate == "postcondition") {
16-        actionToDo = findNextActionToDo()
17-            naturalContent = actionToDo.Activity.goal
18-            perf = INFORM
19-            FipaSLContent = "((iota ?actionToDo.name (postcondition
20-                                    ?actionToDo.name ?iota.object)))"
21-      } else {
22-        naturalContent = "I do not understand what you want"
23-        perf = NOT_UNDERSTOOD
24-      }
25-    }
26-  } else {
27-    naturalContent = "I do not understand what you want"
28-    perf = NOT_UNDERSTOOD
29-  }
30- } else {
```

```
31-   naturalContent = "I do not understand what you want"
32-   perf = NOT_UNDERSTOOD
33- }
```

In our work, we handled inquiring about the description of an action whenever the "*action*" operator is detected. When the user asks about the next desired action, the agent representing the user uses the "*action*" expression in the FIPA-SL content of the FIPA-ACL message that it sends to another agent (Algorithm 5). If the procedure is not finished, the agent prepares the attributes of the COMMUNICATIONACTION to inform the user-agent with the description property of the desired action (Algorithm 5).

Algorithm 5: Inquiring about actions

```
1- if (result.isAction){
2-   actionToDo = findNextActionToDo() //desired action in procedure
3-   value = actionToDo.Description //description stereotype of action
4-   performative = INFORM   //performative
5-   naturalContent = value
6-   FipaSLContent = "((action actionToDo.performer (actionToDo)))"
7- }
```

While managing the received messages in the COMMUNICATIONBEHAVIOR, we ensure that the agent checks for the previously parsed messages in order to give accurate replies. Therefore, the agent can confirm if the consecutively asked questions are valid. For example, when the user asks about the position of an entity and then for a post condition for the position, the agent replies with a message using the "NOT_UNDERSTOOD" performative to notify the user about this fault.

As soon as the context of the reply is specified for the agent, we instantiate a COMMUNICATIONACTION (3.3.2) by setting its attributes with the obtained values (Algorithm 6). We send this action to the inquiring agent in order to be informed with the reply and to plan for a communicative intention.

Algorithm 6: Instantiating Communication Action in manageQueryRef

```
1- if (FIPAmsg.performative == "QUERY_REF"){
```

```
2-  FIPASLParserResult result = parseFipaSL(FIPAmsg.content)
3-  if (result.isIota){
4-     //manage iota
5-  } else if (result.isAction){
6-     //manage action
7-  } else {
8-     // not understood
9-  }
10-  CommunicationAction ca = new CommunicationAction()
11-  ca.performative = p
12-  ca.naturalContent = n
13-  ca.FipaSLContent = FipaSLContent
14- }
```

When the performative of the received message is “INFORM”, the agent parses its FIPA-SL content and updates its knowledge base with the provided information (Algorithm 7). Upon receiving a FIPA-ACL message with the “INFORM” performative and the “*iota*” operator (isIotaEqual), the agent parses its contents to update its own knowledge base with the value of the entity slot if recognized, or about the user whenever she/he starts/accomplish an action in the procedural behavior (Algorithm 7).

To update the knowledge base with the received slot value of the specified entity, the agent checks for this entity among the entities of the ENVIRONMENT in its knowledge base. If the entity is recognized in its knowledge base, the agent updates its knowledge base with the assigned slot value (Algorithm 7).

Algorithm 7: manageInform

```
1- FIPASLParserResult result = parseFipaSL(FIPAmsg.content)
2- entityName = result.iota.entity
3- slotName = result.iota.slot
4- value = result.iota.value
```

```
5- if (result.isIotaEqual){
6-   KnowledgeBase = Agent.KnowledgeBase
7-   entities = KnowledgeBase.getEntities()
8-   foreach (entity in entities) {
9-    if (entity.name == entityName) {
10-      foreach (slot in entity.Slots) {
11-        if (slot.Value.name == slotName) {
12-          slot.Value = value
13-        }
14-      }
15-    }
16-  }
17- }
```

In the following section (3.3.4), we outline how we can generically determine the dialogue contents of the verbal communicative acts of the user and how the agents can recognize these inquiries and parse them successfully. While in section 0, we will explain how the communication actions exchanged by the agents are instantiated by an ECA to manage the natural dialogue.

3.3.4 Taxonomy of Questions

In a virtual environment implemented through our model, the user can ask different types of questions. Therefore, the scope of expected questions has to be defined for the agents. The agents can parse received inquiries from the user, review their knowledge base, and build suitable replies.

The questions of the user that can be foreseen are mainly about the properties of the agents they are interacting with, the entities they should manipulate, and the actions they should perform. Using the communication behavior, the agents respond to all inquiries of the user even when the received questions are not correctly formulated or are out of the scope.

Here is a list of questions with their properties that the agents in MASCARET have in their knowledge base in order to properly reply to the user:

Table III. Expected questions from the user and the necessary resources in MASCARET to reply

Targets in Mascaret	Sources in Mascaret	Questions Styles
Questions about properties of a certain entity		
Entity Properties: Slots, Position, Sound, Animation, Parent, Subclasses…	Agent → Knowledgebase → Environment → InstanceSpecification → Entity → Slots	-What is *EntityName*? -What can I do with *EntityName*? -What are the features of *EntityName*? -What is the function of *EntityName*? -What is the *Slot* of *EntityName*? -Where is *EntityName*? -Which one is *EntityName*? -How would you describe *EntityName*? -How would you explain *EntityName*? -How many *EntityName* we have? -Can you define *EntityName*? -Can you illustrate *EntityName*?
The Agent is a specialization of Entity. Here are the generic questions about agents:		
		-What is *AgentName*? -What can *AgentName* do? -Where is *AgentName*? -Who is *AgentName*?

Targets in Mascaret	Sources in Mascaret	Questions Styles
Questions about properties of behaviors and procedures		
Agent Behaviors: Procedural Execution, Activity, Resource, Role…	Agent → AgentBehavior	-What should I do now? -What can I do now? -What can I do in order to …? -What should I use to *ActionName EntityName*? -Who? -Who has to ActionName EntityName? -How? -How can I ActionName EntityName? -Can I ActionName EntityName? -Why? -Why should I ActionName EntityName? -Why do I have to *ActionName EntityName*? -When can I ActionName EntityName? -When should I ActionName EntityName?

3.4 SAIBA Integration

The different behaviors (desires) of the agents generates (organizes) the execution of actions (intentions). Some of those actions are communication actions (communication intentions). We explained in the previous section how those communication actions are executed in the context of agent communication (in the sense of FIPA).

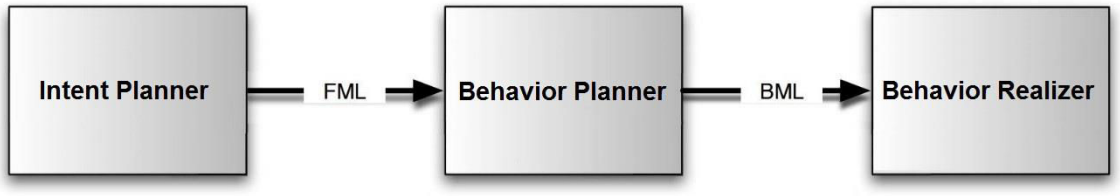

Figure 46. Modules of SAIBA framework

An agent can be an embodied agent. To do that, we embody each agent meant to interact with the user through an ECA which is displayed in the virtual environment. In MASCARET, the embodied agents are defined using the EMBODIEDAGENT class that inherits the AGENT class and adds a body (Figure 47). If the agent is embodied, the communication intentions are added to the agent, and managed by the SAIBA compliant platform. The fact that the communication intentions are added to the agent (by a behavior) permits us to enrich the communication intention with some internal property that an agent can have (but that we did not manage here) such as emotions.

To integrate SAIBA compliant ECAs in MASCARET, we implement two modules of the SAIBA framework [90] through an interface for the Behavior Planner and an interface for the Behavior Realizer (Figure 46).

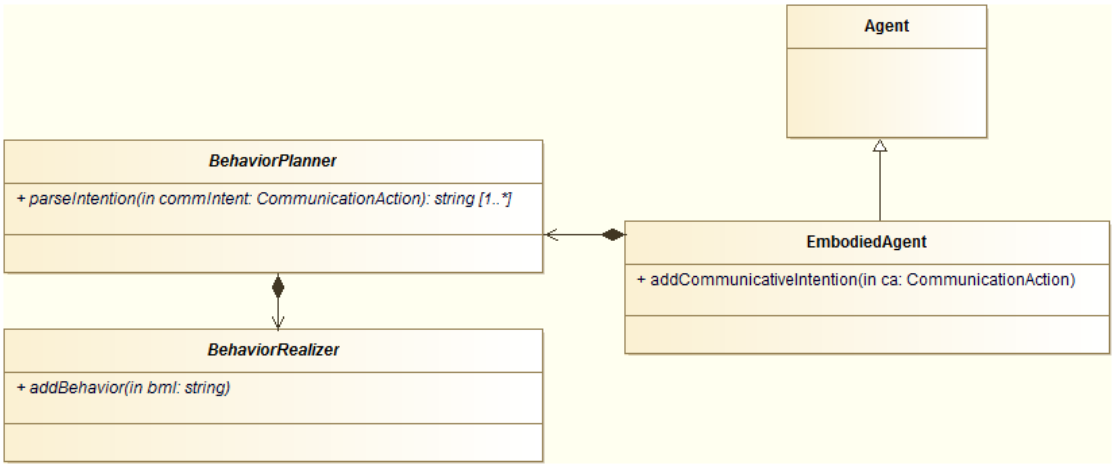

Figure 47. Implementing SAIBA modules for embodied agents

The EMBODIEDAGENT plans for the communicative intentions and represents them through instances of the COMMUNICATIONACTION (see section 3.3.2.2) (Figure 47). The intentions of the agents are prepared in the BDI module that we implemented in MASCARET (see section 3.1). We specify the intentions in the EMBODIEDAGENT since at this level it is only for communication.

3.4.1 Implementation of Behavior Planner and Behavior Realizer interfaces

To connect MASCARET to the two modules of SAIBA (Behavior Planner and Behavior Realizer) that are implemented in ECA platforms, we added two classes in MASCARET (the BEHAVIORPLANNER and the BEHAVIORREALIZER classes) to represent the two modules of SAIBA framework (Figure 47). The BEHAVIORPLANNER is a class with the *parseIntention(CommunicationAction ca)* method. This method has a default generic implementation that just initialize the list of BML strings, but this method has to be overridden by the concrete behavior planner class (see in next section) in the 3D engine. The EMBODIEDAGENT has an association with the BEHAVIORPLANNER (Figure 47) to code the communicative behaviors in a BML format string and sends it to the associated BEHAVIORREALIZER. The BEHAVIORREALIZER class has an abstract method *addBehavior(string bml)* that has to be overridden in the behavior realizer concrete class in the 3D engine. These concrete classes manages the connection with the "Behavior Planner" and "Behavior Realizer" modules of the integrated ECA systems in order to transmit the planned communicative behaviors. In Figure 48, we show the sequence chart of the represented SAIBA modules in our model.

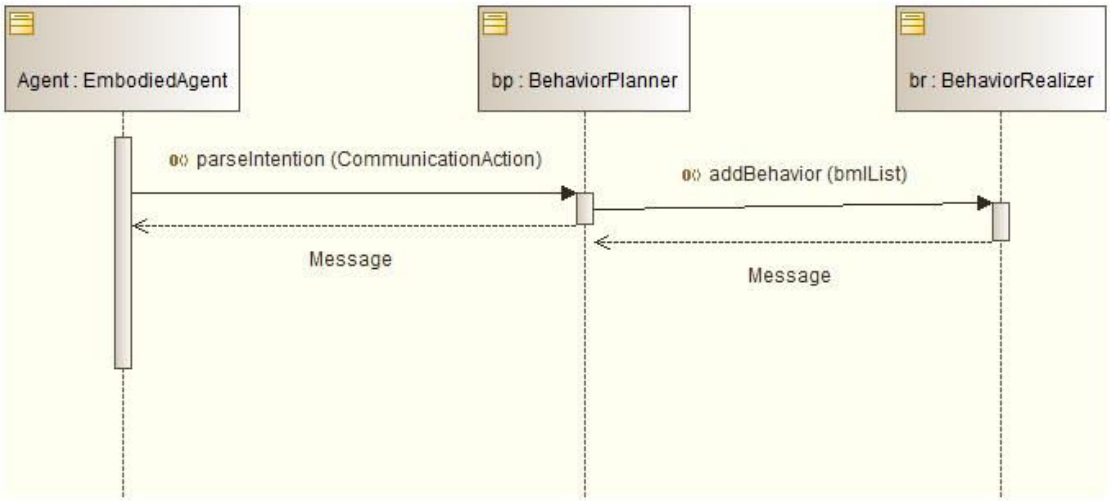

Figure 48. The sequence chart of the represented SAIBA modules in our model

3.4.2 Integrating ECA platforms

We used the interface engine to integrate different ECAs in the virtual environment. The concrete classes of these interfaces implement the abstract methods of the BEHAVIORPLANNER and BEHAVIORREALIZER of MASCARET.

Among the SAIBA compliant ECA platforms, we had to connect at the "Behavior Planner" level of the ECA (e.g. Greta). This ECA parses the received behavioral signals

in order to realize the communication behaviors. While for other ECAs (e.g. Virtual Human Toolkit and MARC) we had to connect at the “Behavior Realizer” level.

3.4.2.1 ECA systems that only provide a Behavior Realizer

Virtual Human Toolkit and MARC provides a “Behavior Realizer”, so they can receive a list of behavioral signals, coded in BML language, and generate the corresponding animation on the virtual agent representation they provide in a player.

In the Unity3D interface engine, we developed the UNITYVHBEHAVIORPLANNER, UNITYVHBEHAVIORREALIZER, UNITYMARCBEHAVIORPLANNER and UNITYMARCBEHAVIORREALIZER concrete classes that inherits the abstract classes of MASCARET in order to plan for behavioral signals that are sent to the “Behavior Realizer” of the ECA which realizes them.

MARC's BML is a hybrid version between BML 0.9 and BML 1.0. For this reason, MARC researchers recommend using the BML visual editor that is included within the MARC-toolkit in order to generate BML messages that are valid in MARC. Therefore, we developed the UNITYMARCBEHAVIORPLANNER to generate similar BML scripts. While the Virtual Human Toolkit accepts the standard versions of BML (0.9 and 1.0) that can also be generated through the UNITYVHBEHAVIORPLANNER that we developed. Within these two concrete classes, and based on the properties of the communicative intentions (performative, sender, receivers, natural content, resources, emotions…), we override the *parseIntention* method to parse the received communicative intentions and code the behavioral signals in the relative BML language messages of MARC and Virtual Human Toolkit. In other words, we check for the “performative” property of the communicative intention in these classes, and for example, when the “inform” function is detected, a speech behavioral signal is set based on other properties like the “natural content”.

We developed the *parseIntention* method in the UNITYVHBEHAVIORPLANNER to generate the BML script, which is compatible with the VHToolkit, according to the performative function that is specified in the communication action received. For instance, when the “INFORM” performative is detected, we use the speech behavior to inform the user with the natural content, and if the resource is also specified in the communication action, the animation for indicating this resource is selected. Here is the simple scenario that we implemented:

```
parseIntention (CommunicationAction action) {
string rsrc
if (action.resources.count > 0) rsrc = action.resources[0].name
string msg
msg = "<?xml version="1.0" encoding="UTF-8"?>
            <act>
              <participant id="agent.name" role="actor"/>
                <bml>"
                 if (action.performative == "INFORM") {
                    <speech id="sp1" type="application/ssml+xml">
                       action.naturalContent
                    </speech>"
                 }
if (rsrc) {
   msg += "<animation name="IndicateResource" target="rsrc"/>
} else if (action.performative == "NOT_UNDERSTOOD") {
   msg += "<animation name="ShowNegative"/>"
} else {
   msg += "<animation name="OfferWithBothHands"/>
}
message+="</bml></act>"
List<string> bmlList
bmlList.Add(msg)
return bmlList
}
```

We send these messages (*bmlList*) to the concrete classes (UNITYVHBEHAVIORREALIZER and UNITYMARCBEHAVIORREALIZER) using the *addBehavior* method of the BEHAVIORREALIZER class of MASCARET that we override in these classes. We developed these classes in order to transmit the relative multimodal behavioral signals to the "Behavior Realizer" of the corresponding ECA in the appropriate BML language. We communicate with the ECA of the Virtual Human Toolkit through the UNITYVHBEHAVIORREALIZER by sending ActiveMQ[8] messages using the VH Messaging library[9]. Whereas, we built a UDP connection with MARC through the UNITYMARCBEHAVIORREALIZER to transmit the multimodal behavioral signals to the "Behavior Realizer" of MARC. The "Behavior Realizers" of these ECAs can generate the agent animation from the multimodal signals received in order to be realized.

3.4.2.2 An ECA system that provides a Behavior Planner and a Behavior Realizer

Other ECA platforms, like Greta, provide a "Behavior Planner" to plan for multimodal signals from the received communicative intentions, and a "Behavior Realizer" to generate the agent's animations from these signals.

Using our model, [97] connected Greta to an EMBODIEDAGENT in MASCARET. To do that, they developed the UNITYGRETABEHAVIORPLANNER concrete class that implements the abstract method (*parseIntention*) of the BEHAVIORPLANNER classes in MASCARET to transmit the communicative intentions to Greta.

Moreover, Greta has its own world representation. In order for Greta to realize communication behaviors in the virtual environment, a link was created between the virtual environment and the environment in the Greta module [97]. For instance, when Greta has to show objects, we have to send the position of the agent and the positions of all the objects involved in the communication action.

The communicative intention for Greta are planned in MASCARET through the COMMUNICATIONBEHAVIOR of agents. It has to be assigned to the EMBODIEDAGENT that is connected to Greta using the *addCommunicativeIntention* method. We developed the UNITYGRETABEHAVIORPLANNER concrete class in the 3D engine that inherits the

[8] https://confluence.ict.usc.edu/display/VHTK/3rd+Party+Software (accessed November, 2017)

[9] https://confluence.ict.usc.edu/display/VHTK/VHMsg (accessed November, 2017)

BEHAVIORPLANNER. We override the *parseIntention* method to parse the received communication actions and to transmit them to Greta.

When the UNITYGRETABEHAVIORPLANNER receives a communication action, it checks the properties of this action (performative, sender, receivers, natural content, resources, emotions…) to code the communicative intentions in FML-APML, and sends them to the "Behavior Planner" of Greta. For example, when the "INFORM" performative function is detected, the UNITYGRETABEHAVIORPLANNER can formulate the following message, which has to be sent to Greta in order to communicate with the user:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<fml-apml>
 <bml>
  <speech id="s1" language="english" voice="openmary" type="SAPI4"
   text=""><description level="1" type="gretabml"><reference>
   tmp/from-fml-apml.pho</reference></description>
    <tm id="tm1"/>
        Natural Content
    <tm id="tm2"/>
  </speech>
 </bml>
 <fml>
  <performative id="p2" type="inform" start="s1:tm1" end="s1:tm2"
  importance="1.0"/>
 </fml>
</fml-apml>
```

We established a link between the properties of the received communication action, which are set according to the FIPA-ACL messages received by the agents in MASCARET (as we mentioned earlier), and the elements of the FML-APML script that has to be generated in the UNITYGRETABEHAVIORPLANNER. Therefore, in the

UNITYGRETABEHAVIORPLANNER, we depend on the properties of the received communication actions (mainly the Performative) to code the communicative intentions and the natural text the agent of Greta has to utter. For example, the above FML-APML script could be formed when the agent in MASCARET receives the following FIPA-ACL message:

```
(inform
 :sender (agent-identifier :name Agent)
 :receiver (set (agent-identifier :name UserAgent))
 :content ((= (iota ?description (slot ?description ?EntityName)) NaturalContent))
 :language FIPA-SL
)
```

As seen in the example, we set the "inform" performative and the "speech" tag in the FML-APML script, according to the "performative" and the "naturalContent" properties of the communication action that are set in MASCARET based on the FIPA-ACL message received by the agent.

We show in Table IV the performative functions of FIPA and FML that we linked to facilitate coding the communicative intentions of Greta.

Table IV. Relating FIPA-performatives and FML-performatives

FIPA-Performatives	FML-Performatives
Accept Proposal	accept, approve
Agree	Agree
Call for Proposal	order, incite
Confirm	Confirm
Disconfirm	Disagree
Inform	Inform
Propagate	Announce
Propose	propose, suggest

Query Ref	question, ask
Refuse	Refuse
Reject Proposal	Criticize
Request	Request

In the next chapter (Chapter 4 Application), we show how we instantiate the generic architecture of our model (Figure 49) in order to implement a pedagogical situation in a VLE that involves tutor agents and the user. In our application, we consider the ECAs that are represented in the virtual environment as tutors that can assist the user to perform the domain procedure. The user interacts with the system to submit inquiries and requests.

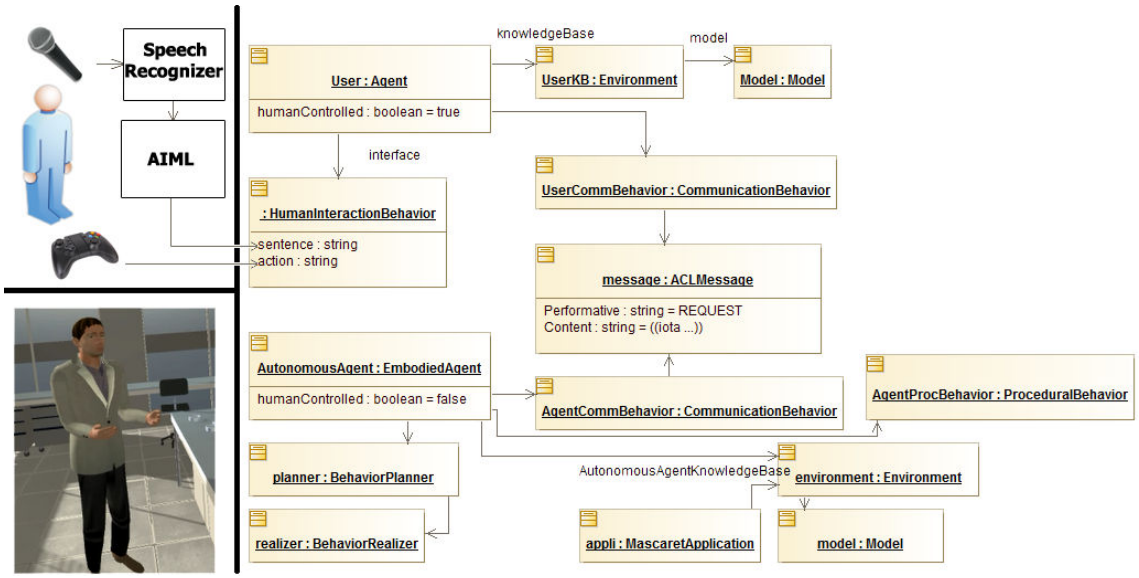

Figure 49. Instantiation of the global architecture by a user and an agent

3.4.2.3 Natural language from user

Throughout the execution of required procedure, the user aspire for guidance and assistance from the collaborative ECA. To forward requests and inquiries, the user can submit natural language text or simply speak to the ECA. They can also show facial expressions and perform gestures to interact with the ECA and perform actions. The agents should perceive the user and detect her/his multimodal signals and communicative acts.

As we mentioned, agents can only communicate through an agent communication language (see section 3.3.2.1). Therefore, the user's communicative acts have to be translated before being delivered to the agents. As explained in section 3.1, MASCARET

represents the user as an instance of the class AGENT (the user-agent) in order to communicate with the agents in the virtual environment.

To transmit user's inquiries to the agents in MASCARET, we used two modules to translate these inquiries from natural language into FIPA-ACL messages that the user-agent can send to other agents in MASCARET. To transform the vocal messages of the user into natural language text, we integrated the Intel RealSense voice recognizer SDK [98]. We used the AIML (Artificial Intelligence Markup Language) chatbot [99] to translate the natural language text into FIPA-ACL messages.

The AIML is an XML-compliant language that depends on generic rules that consist of tags, words, spaces and wildcards ("*" and "_") to parse natural language text and generate relative messages (like FIPA-ACL messages). These rules have to be built upon the expected inquiries of the user. For example, when the user inquires about the description of an entity in the virtual environment using the question "What is *EntityName*?", we have to create the following AIML generic pattern to generate the corresponding FIPA-ACL message:

AIML:

```
<category>
        <pattern> What is *</pattern>
        <template>
            ((iota ?description (slot ?description ?<star index="1"/>)))
        </template>
</category>
```

FIPA:

((iota ?description (slot ?description ?*EntityName*)))

In our model, we developed most of the generic patterns to translate the expected questions of the user (listed in section 3.3.4) into FIPA-ACL messages that can be sent to the agents communicating with the user.

As seen above, the agents depend on the COMMUNICATIONBEHAVIOR to communicate with each other and exchange information. When the user-agent sends a FIPA-ACL message to other agents using its COMMUNICATIONBEHAVIOR, the agent that receives

this message parses it and checks its knowledge base in order to prepare for a response. Subsequently, this agent formulates the response with a new FIPA-ACL message and sends it to the user-agent using its COMMUNICATIONBEHAVIOR. This message is then transmitted to the user through the ECA.

Here is a communication scenario that could take place in the virtual environment between the user and the agents:

User's question:

What is EntityName?

AIML pattern:

```
<category>
  <pattern>What is *</pattern>
  <template>
      ((iota ?description (slot ?description ?<star index="1"/>)))
  </template>
</category>
```

The generated **FIPA-ACL** message that the user-agent sends to other agents in MASCARET:

```
(query-ref
 :sender (agent-identifier :name User-Agent)
 :receiver (set (agent-identifier :name Agent))
 :content ((iota ?description (slot ?description ?EntityName)))
 :language FIPA-SL
)
```

When the agent receives the FIPA-ACL message, it checks its knowledge base and the procedural scenario to obtain the requested information. Different responses can be generated:

Case 1: The agent succeeds to obtain the description of the specified entity from its own knowledge base and transmits it to the user-agent.

Case 2: If the agent does not recognize the specified entity, it forwards a similar FIPA-ACL message to other agents. If one of the agents succeeds to acquire the desired information, it sends a response to the main agent through a FIPA-ACL message. The main agent can then forwards the response to the user-agent:

```
(inform
 :sender (agent-identifier :name Agent)
 :receiver (set (agent-identifier :name User-Agent))
 :content ((= (iota ?description (slot ?description ?EntityName)) descriptionValue))
 :language FIPA-SL
)
```

Case 3: Neither the solicited agent nor the other contacted agents succeed to obtain the description of the specified entity. The solicited agent uses the "DISCONFIRM" performative function to send a FIPA-ACL message to the user-agent to indicate that the requested information cannot be obtained:

```
(disconfirm
 :sender (agent-identifier :name Agent)
 :receiver (set (agent-identifier :name User-Agent))
 :content ((iota ?description (slot ?description ?EntityName)))
 :language FIPA-SL
)
```

In addition to the verbal communicative acts, ECAs can perform non-verbal signals, such as pointing to an object and lips synchronization. After determining the communicative intentions, our SAIBA interface takes care of the virtual agent animation (as explained in section 0).

4 APPLICATION

In the previous chapter, we presented the structure of our proposed model and the interactive capabilities that can be realized by the agents and the user. We focused on showing the knowledge components that can be gathered by the agents, the actions and behaviors they can apply, the activities which the user can perform, and the tools and frameworks that we integrated in order to provide natural interaction between the agents and the user.

In this chapter we show how our model is used to develop VLEs to tutor the user applying procedures in a specific domain. More precisely, we show how we used our model to build a VLE on the biomedical domain for training on a blood analysis instrument. The aim of the developed application is to tutor the user on a pre-operative blood analysis procedure. The environment is a biomedical laboratory including necessary products [100].

4.1 Technical Architecture

In this section, we present the technical architecture to apply our proposed model. We show how a real pedagogical scenario with tutoring objectives can be implemented using our model. The VLE is implemented using Unity3d engine. In the virtual environment, we added the virtual entities that have to be manipulated while applying the actions of the procedural scenario but also some elements for decorations. In addition, we integrated ECAs in order to interact with the user in a pedagogical objective (Figure 51). According to the pedagogical scenario, various roles can be assigned to these agents while tutoring the user (detailed in Section 2.1.3).

We first present the composition of the virtual environment (section 4.1.1) and then we present the domain model (section 4.1.3).

4.1.1 The virtual environment

The virtual environment of the blood analysis procedure is composed of several products and tools (tubes, scanner, basket, paillasse, buttons, rack...) that the user has to manipulate while executing the actions of the procedure. Among these entities, there are the automaton (Figure 50-a) and a set of chemical and blood tubes required to prepare the reagents (Figure 50-b).

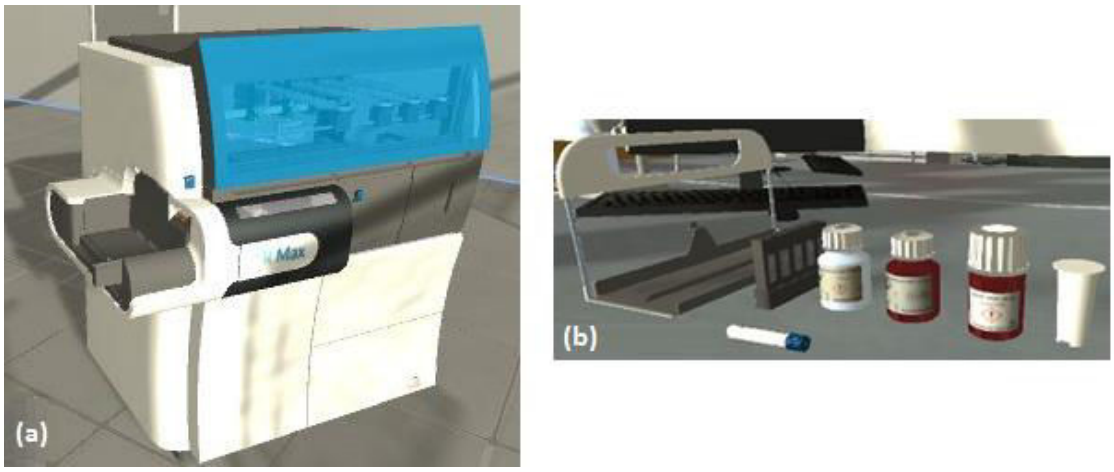

Figure 50. The automaton and products for reagents preparation

4.1.2 Integrating the ECAs

As we described in our model, we made it easy to integrate ECA platforms that are compliant with the SAIBA framework. Three well known ECA platforms have been integrated: Greta, Virtual Human Toolkit and MARC. One embodied agent of each platform can be seen in Figure 51. These embodied agents can represent the tutors in the virtual environment and they can communicate naturally with the user.

We integrate these ECA platforms in our environment according to their particular specifications. For instance, as shown in Figure 51, the ECA of MARC is represented in a separate window with a similar environment. It could not be added into our environment since it requires defining its own environment using its own specific 3D engine. This is not very convenient, we then used only the two agents that can be integrated in the Unity3D: VHToolkit and Greta.

After implementing the concrete classes that are responsible for sending the communicative intentions and behaviors to these ECAs, we assign a particular tutoring role to each ECA (detailed in Section 4.1.2).

Moreover, the user can address any of the embodied agents in the environment by saying the name of the intended agent while interacting with the system (explained in the implemented scenario in Section 4.3).

Figure 51. The integration of various ECA platforms

4.1.3 Building the domain model

All the necessary properties (name, position…) and operations (*Take*(object), *Press*(object)…) of the entities that are set in the virtual environment are defined in the domain model. This domain model is defined in MASCARET through the UML classes and behavioral models (Figure 52) in order to be used by the ECA as a knowledge base to execute the pedagogical scenario.

We used the Modelio[10] UML modeler to build the appropriate class models, activity diagrams and state machines.

[10] https://www.modelio.org/ (accessed November, 2017)

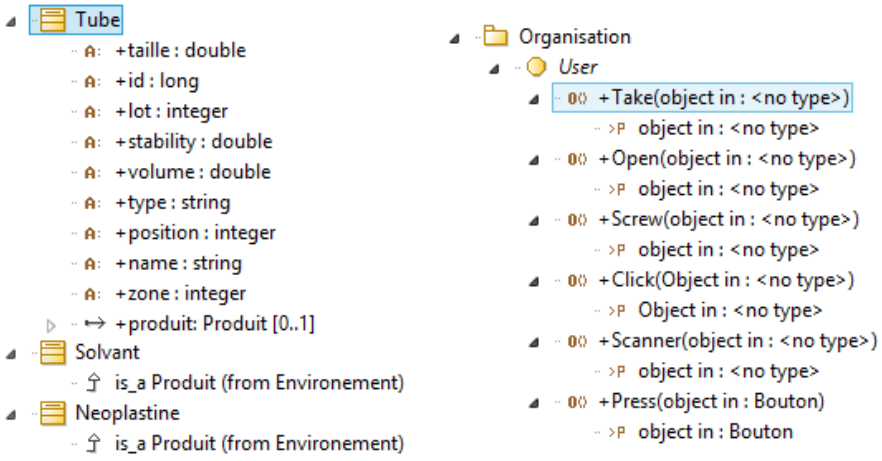

Figure 52. Properties and operations of entities

In this section, we take the example of the blood analysis procedure. This procedure is composed of a sequence of 38 actions, such as holding and opening tubes, using the automaton, and scanning mixed reagents (Appendix 3).

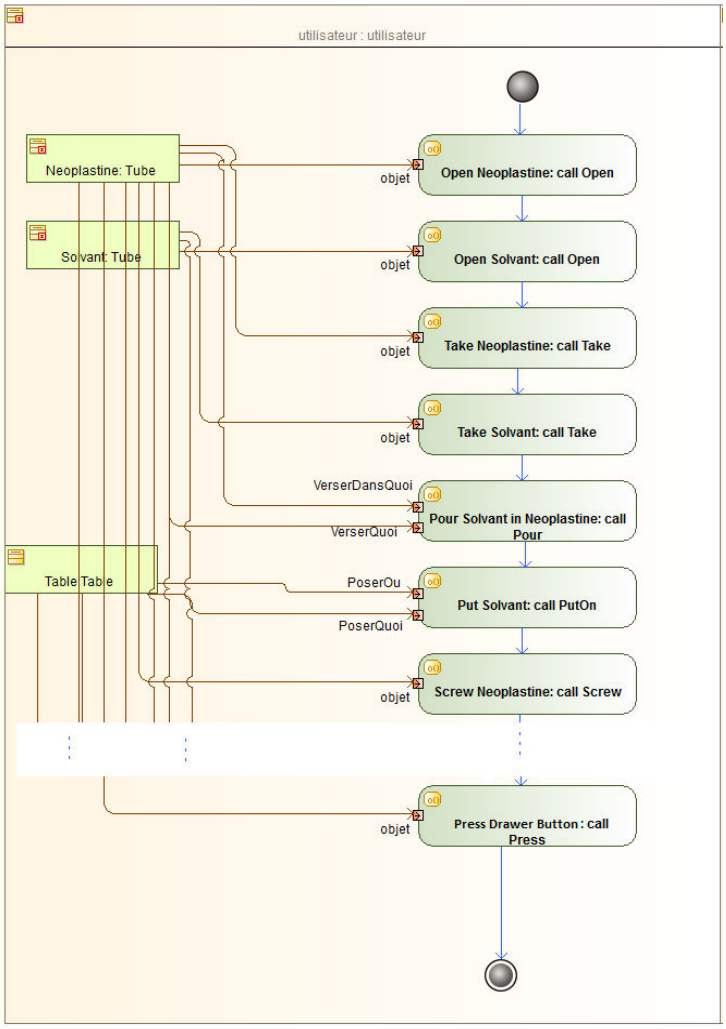

Figure 53. Extract of blood analysis procedure

We defined this procedure in an activity diagram by setting the initial, the final and the domain action nodes. For each action, we had to set its name, call the appropriate operation and link it to one or more defined objects (Figure 53). The user should manipulate these objects in order to execute these actions. For example, to mix the two chemical products, “Neoplastine” and “Solvent”, the user has to find the tubes of “Neoplastine” and “Solvent” in the virtual environment and consecutively perform the “Take” and “Open” actions in order to prepare the reagents.

The embodied agents in our system can inform the user about the actions she/he should perform and about the objects she/he should manipulate. We defined a pedagogical scenario with two tutoring roles, designated by the pedagogical expert. In this section, we take the example of a pedagogical scenario for a user who realises the procedure for the first time. In this case, one agent has to explain the goal of the global procedure (and its subparts) and the other has to give the instructions (what action to do) and to show the object to manipulate. This pedagogical scenario comes from our collaboration with colleagues in cognitive psychology. As shown in Figure 54, we added a set of pedagogical actions to the procedural scenario in the UML modeller (Modelio) and connected them to the actions from the domain procedure that the user has to perform. The pedagogical actions are distributed within the pedagogical scenario under the two roles of the tutor agents (Role 1-Formateur and Role 2-FR).

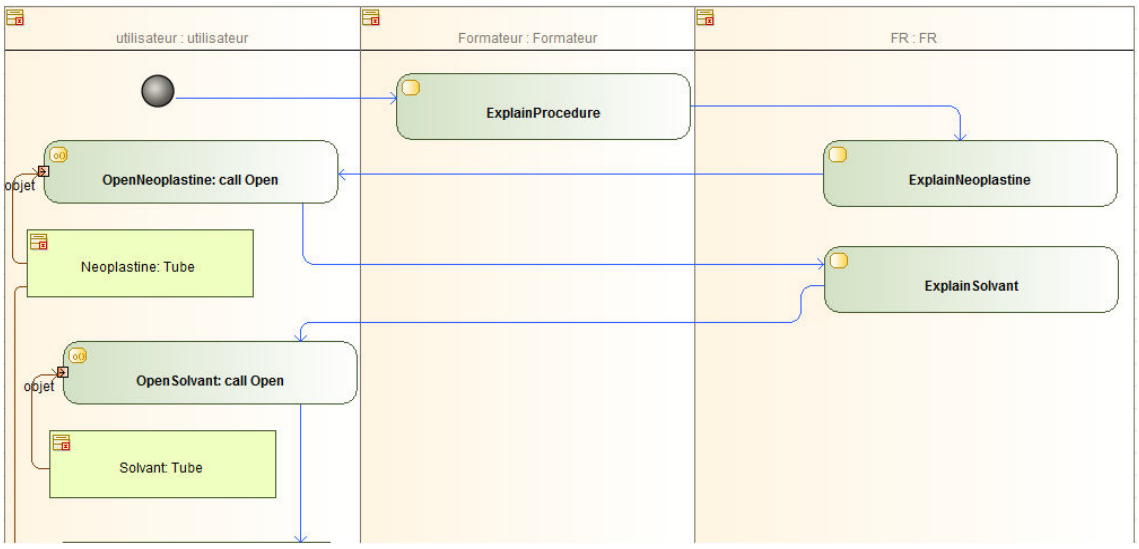

Figure 54. Building a pedagogical scenario with necessary roles

We defined a “*Description*” property for each domain action. This is done by the domain expert to generate the documentation but it is also used in our application by a

tutor agent to present the instructions explicitly to the user. This property can easily be updated by the instructor while developing the pedagogical scenario (Figure 55).

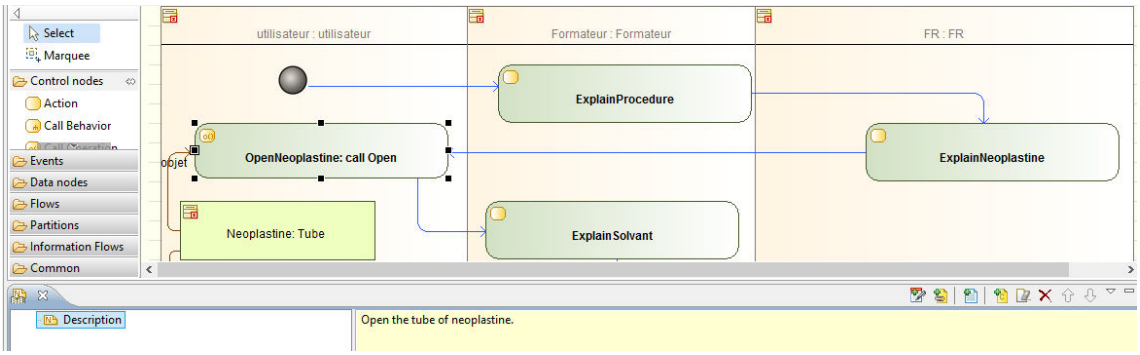

Figure 55. Description property to explain domain actions

To define the agent communicative actions, we added the following fundamental properties to every pedagogical action that refers to a communication with the learner (Figure 56):

1- *Performative*: the performative function,
2- *Receivers*: the receivers of the message,
3- *Natural*: the natural language text to communicate with the user,
4- *FIPA*: the FIPA-SL script of the communicative action,
5- *Resources*: the virtual objects to be manipulated.

For example, in the first pedagogical action in Figure 56, the agent has to inform the user about starting the procedure, we defined an agent communicative action by specifying the "user" as the *Receivers*, the "inform" as the *Performative* function and a natural text "We will start to prepare the reagents" in the *Natural* property to inform the user about starting the procedure (Figure 56). When we needed to communicate with a tutor agent to request an action (like informing the user about the description of a domain action), we had to set the *Performative* property and the FIPA-SL content "((action user (NEXT)))" in the *FIPA* property (Figure 57).

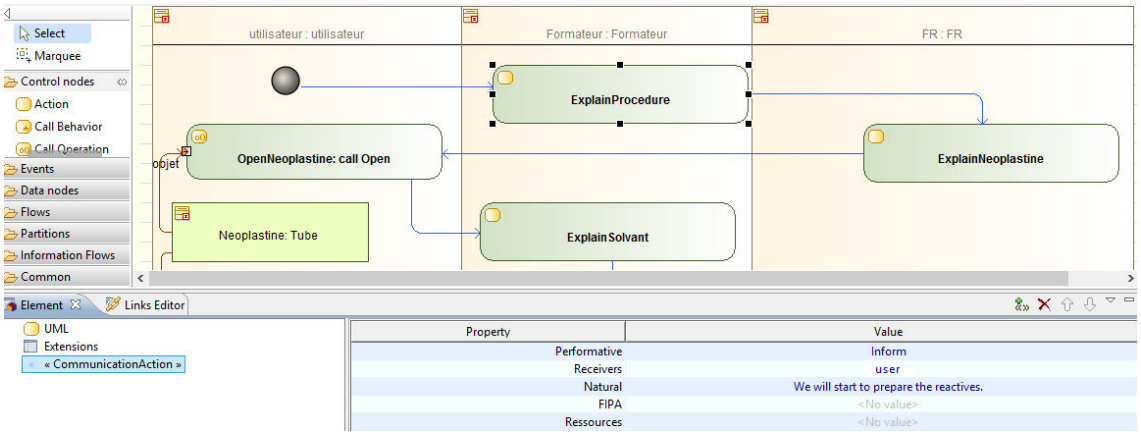

Figure 56. First pedagogical action with necessary properties

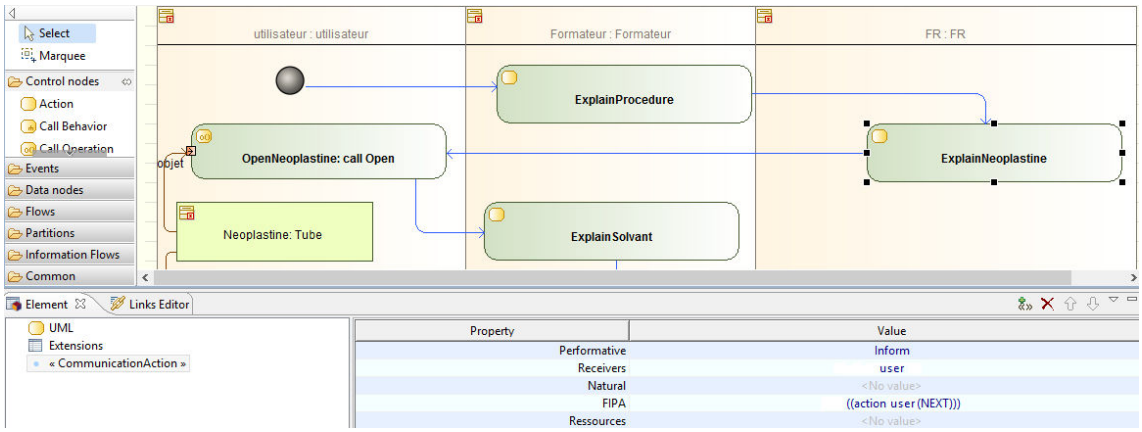

Figure 57. Defining actions for the tutor agents in the pedagogical scenario

4.1.4 Importing MASCARET and the defined models

To integrate the MASCARET metamodel in the application in order to execute the domain and the pedagogical models, we imported MASCARET library using a DLL (Dynamic-link Library) file (Mascaret.dll) into our application in the Unity3D engine.

After building the class models and the behavioral models (domain model and pedagogical scenario) in Modelio, we exported them as an XMI (XML Metadata Interchange) file. To consider these models in our application, we imported this XMI file in the interface engine (in the StreamingAssets directory for Unity3D). We had to apply a similar process (exporting and importing the XMI file) whenever we modify any component in the class and behavioral models.

Moreover, we had to represent the virtual environment to MASCARET in order to consider it and use its entities. Therefore, we built an XML (eXtensible Markup Language) file (environment.xml) to outline the structure of the environment by representing all the virtual entities in the environment, their properties (such as names,

positions, shapes…) and the affectation of ECA to roles. We put this XML file in the StreamingAssets of Unity3D and refer it in MASCARET.

4.1.5 Communication methods for the user

In addition to the specified peripherals for interacting with the system, the user can vocally communicate with the agents in the virtual environment. We used the Intel RealSense SDK to implement the voice recognition module in the Unity3D engine. It recognizes the natural language speech and converts it to simple text in order to submit the inquiries of the user to the tutor agents. The RealSense SDK depends on a library with dictionary words and phrases to recognize the contents of the vocal messages. However, the vocabulary words of the concerned domain can be added to this library using a text file. The user can also use the text field displayed in the interface of the application in order to communicate with the agents using text messages that can be submitted in this form (Figure 58).

As we mentioned in Chapter 3, the agents in our model use FIPA-ACL messages to communicate with each other to share their knowledge bases and cooperate to execute actions. Therefore, we need to translate the text received from the user into FIPA-ACL messages in order to send the inquiries to the tutor agents. We defined the appropriate AIML generic patterns of questions that the user can ask. The agent representing the user in MASCARET sends these messages to the tutor agents. Parsing rules, which are set in the FipaSL.g4 module in MASCARET, are used to parse the FIPA-ACL messages of these agents.

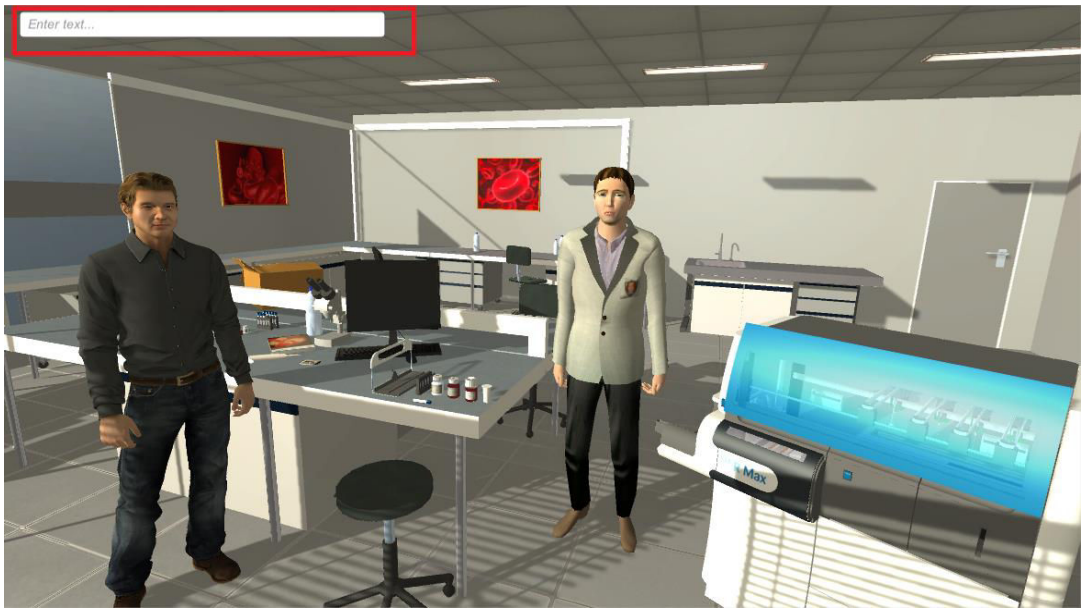

Figure 58. Simple text field to communicate with the agents

4.2 Tutor Behavior

The PROCEDURALBEHAVIOR executes the pedagogical scenario to guide the user in performing the domain scenario. To ensure applying the domain actions of the scenario, the agents in the VLE should be able to monitor the activities of the user including the inquiries that could be submitted. Accordingly, we defined a TUTORBEHAVIOR (Figure 59) that could be executed by the agents in MASCARET that play the role of the tutor.

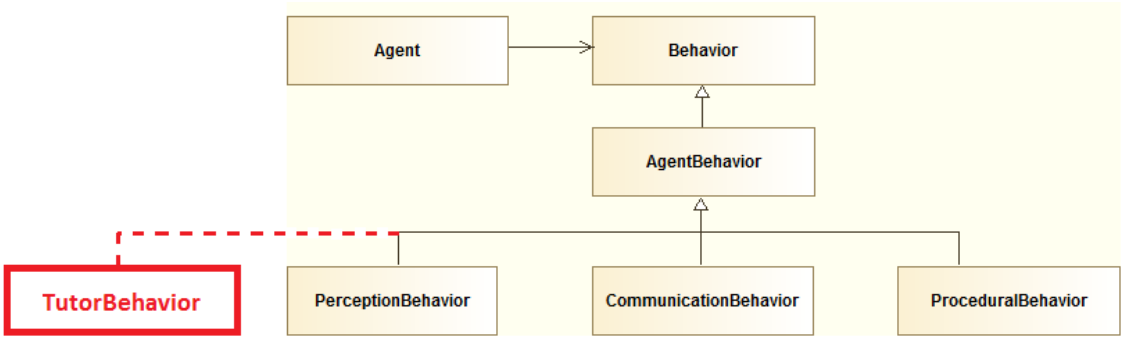

Figure 59. Adding the Tutor Behavior for the agents

The tutor agent follows the pedagogical scenario through the UML activity diagram defined by the domain instructor. This agent executes the pedagogical actions which are connected to the actions of the domain procedure. For example, when a domain action requires applying a behavior on a certain object, we connect a pedagogical action to the domain action in order to explain its description to the learner and to indicate which object should be manipulated. The tutor agent executes the pedagogical actions by communicating with the learner using communication actions.

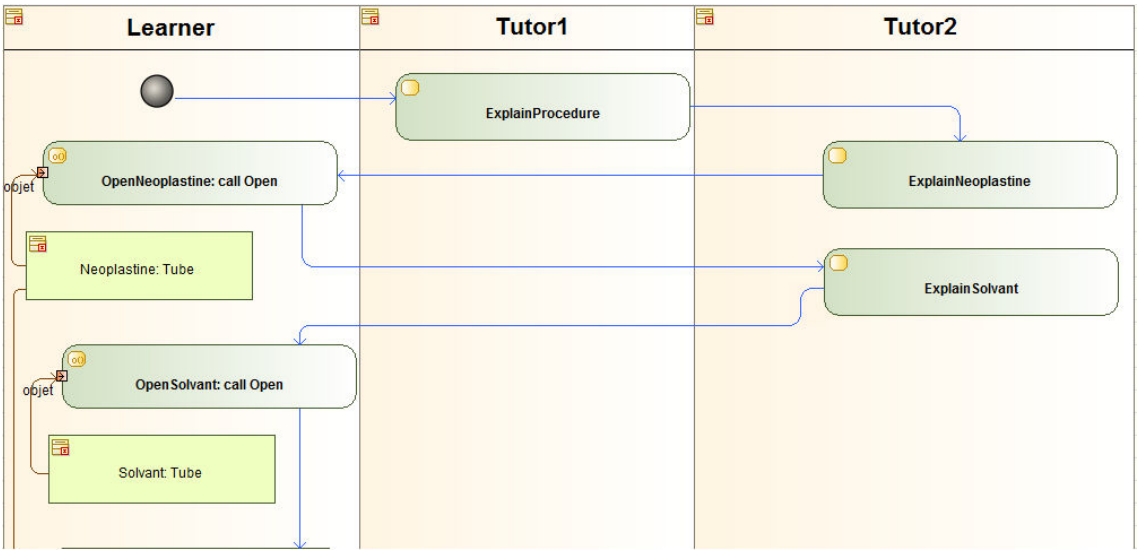

Figure 60. Procedural scenario with pedagogical actions

As an example, Figure 60 shows a procedural scenario with domain actions that are preceded by pedagogical actions to clarify to the learner the required actions to perform and the objects to manipulate. However, the learner can interrupt the domain scenario at any time and asks for a precision or new information (Figure 61). When such event is recognized, the tutor agent has to work on replying to the inquiry.

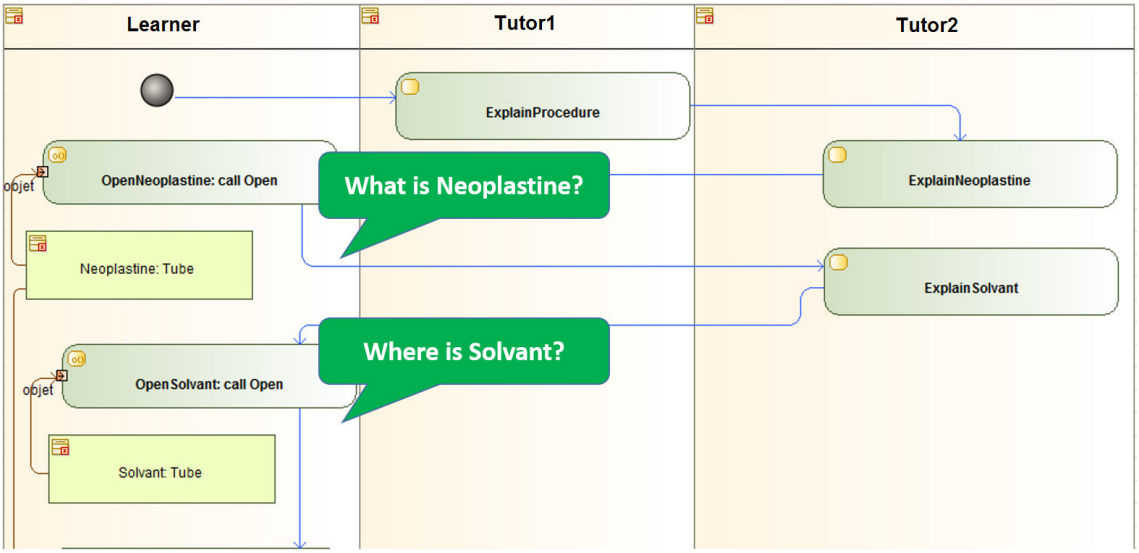

Figure 61. Interrupting the pedagogical scenario of a procedure

In the following example, when the learner needs to ask the agent (named "Mike") about the required action, the learner starts the question by specifying the name of the targeted agent (Mike). We defined the following generic AIML pattern to generate the FIPA-ACL message that is sent to the tutor-agent (Mike) in order to reply:

Text:

Mike what should I do now?

AIML pattern:

```
<category>
        <pattern>* What should I do now</pattern>
        <template>
            <star/>;((action <get name="user"/> (NEXT)))
        </template>
</category>
```

Generated FIPA-ACL message:

Mike;((action User-Agent (NEXT)))

While executing the domain procedure, the tutor-agents monitor the communicative acts performed by the learner. The agent representing the learner uses its COMMUNICATIONBEHAVIOR to forward the FIPA-ACL message, which represent the inquiries of the learner, to the tutor-agent in order to reply. The tutor-agent checks its knowledge base, the procedural scenario and the previously exchanged messages with the agents in order to determine and provide the suitable reply. For instance, when the tutor-agent informs the learner about the goal of an action, the learner might repeatedly ask for the purpose of performing that action. Rather than just repeating the same post-condition to the learner, the tutor-agent refers to the hierarchy of the procedure to provide the learner with a further knowledge about the required action, such as the goal of the activity.

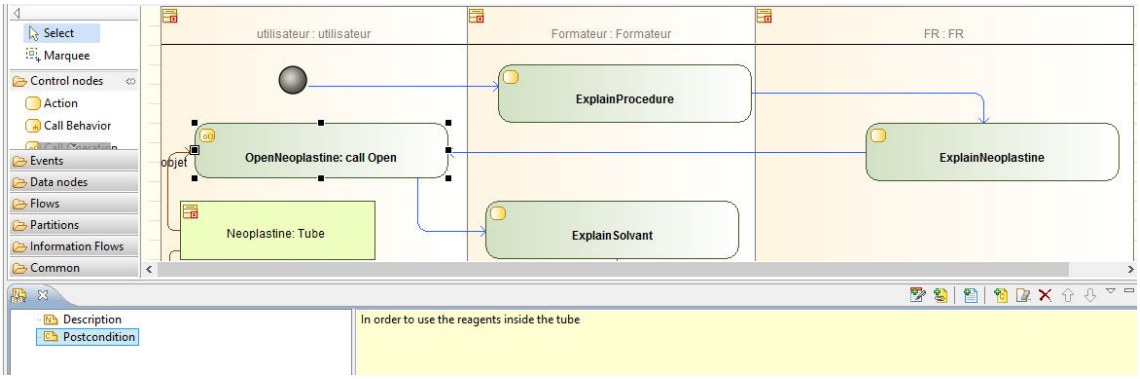

Figure 62. Post conditions of actions

While executing the domain procedure, the tutor agent constantly checks its environment and the running procedure and executes the TUTORBEHAVIOR. It monitors the activities performed by the learner. When the learner performs an action, the tutor-agent compares this action with the required action in the running domain procedure. The tutor-agent starts by checking if the learner manipulated the entity that is connected to the required domain action. If an incorrect entity is manipulated, the tutor-agent prepares a COMMUNICATIONACTION with the "INFORM" performative to inform the learner that a wrong entity is selected using this **Natural-Content**: "This is *wrongObjectName* and not *correctObjectName*".

The learner could manipulate the correct entity in the VE but performs a different action from the desired one in the domain procedure. In this case, the tutor-agent prepares a

COMMUNICATIONACTION to inform the user that this is the correct object but an incorrect action is selected using this **Natural-Content**: "This is *correctObjectName*, but you have to *correctOperationDescription*".

When the learner performs the correct action and manipulates the connected entity to the domain action, a "DONE" action message is sent to the tutor-agent in order to proceed to the next action planned in the domain procedure and it realizes the corresponding pedagogical action. Upon finishing the domain procedure, the tutor-agent informs the learner that the procedure is accomplished.

4.3 Implemented Scenario

We implemented the blood analysis procedure (Figure 53) using the different types of the defined behaviors of the agents. The scenario consists in tutoring the learner on the domain procedure using the corresponding pedagogical scenario (Figure 54). We assigned various tutoring roles to the instances of the three integrated ECA platforms (Virtual Human Toolkit-Mike, Greta-Obadiah and MARC-Simon) to show how different roles can be specified for the agents (Figure 63-1). Mike realizes the first pedagogical action to inform the learner about starting the procedure (Figure 63-2). We defined the communication action of this pedagogical action in the activity diagram of the scenario with the "inform" performative and the natural language text that Mike has to say to the learner through its verbal behaviors (Figure 56). Obadiah executes the second pedagogical action that requires informing the learner about the description of the first domain action (Figure 54, Figure 63-3). As we explained earlier, the FIPA property of the communication action in this pedagogical action indicates getting the description property, which is defined by the domain expert, from the domain action using the appropriate FIPA message (Figure 57).

The learner can interrupt the scenario and ask questions to acquire for additional information. The learner might inquire about the goal of the requested action by asking "Why?" (Figure 63-4). When the learner does not specify the tutor agent that she/he intends to interact with, Obadiah who is the default agent processes the inquiry and replies to the learner with the post-condition of the requested domain action (Figure 63-5). Moreover, the learner might intend to obtain additional information about the action she/he has to perform by consecutively asking "Why?" (Figure 63-6). The tutor agent refers to the hierarchy of the procedure to provide the learner with the activity goal as a further knowledge to that action (Figure 63-7).

The learner can communicate with a particular tutor agent by naming it when she/he formulates the question. For example, when the learner intends to ask Mike about the description of the object that she/he should manipulate, she/he has to start the question with "Mike" (Mike, why?). The learner can vocally communicate with the tutor agent using a microphone, or they can submit their questions using the text field that is displayed in the interface (Figure 63-8). Mike obtains the description of the object from its "Description" property and vocally transmits it to the learner (Figure 63-9).

The location of the appropriate object might be unknown to the learner. She/he could try to use the wrong object to execute the action (Figure 63-10). Since the tutor agents are constantly monitoring the activities of the learner, they can discover that an incorrect object is selected and inform the learner with the committed error (Figure 63-11). The learner can then ask about the position of the object using the "Where is *ObjectName*" phrase and naming the object in the inquiry (Figure 63-12). After getting the orientation of this object from the environment, Obadiah (the default tutor agent) points with his hand to the object that also blinks with red color, and he vocally tells its place (Figure 63-13). Furthermore, when the learner selects an inappropriate action on the right object (Figure 63-14), Obadiah informs the learner about the committed fault after checking the domain procedure (Figure 63-15).

When the learner performs the requested action (Figure 63-16), the corresponding animation, like opening a tube, is realized in the virtual environment (Figure 63-17). The pedagogical scenario automatically proceeds to the following pedagogical action that consists in informing the learner about the next domain action she/he has to perform (Figure 63-18). When the learner performs all the domain actions, Obadiah announces the accomplishment of the procedural scenario with a thankful message (Figure 63-19).

In Chapter 5, we evaluate our model by describing the experiment we conducted on an experiment on a group of participants using a demonstration of the developed application. We apply evaluation techniques and relative statistical tests on the obtained results of the experiment in order to validate the methodologies of our model.

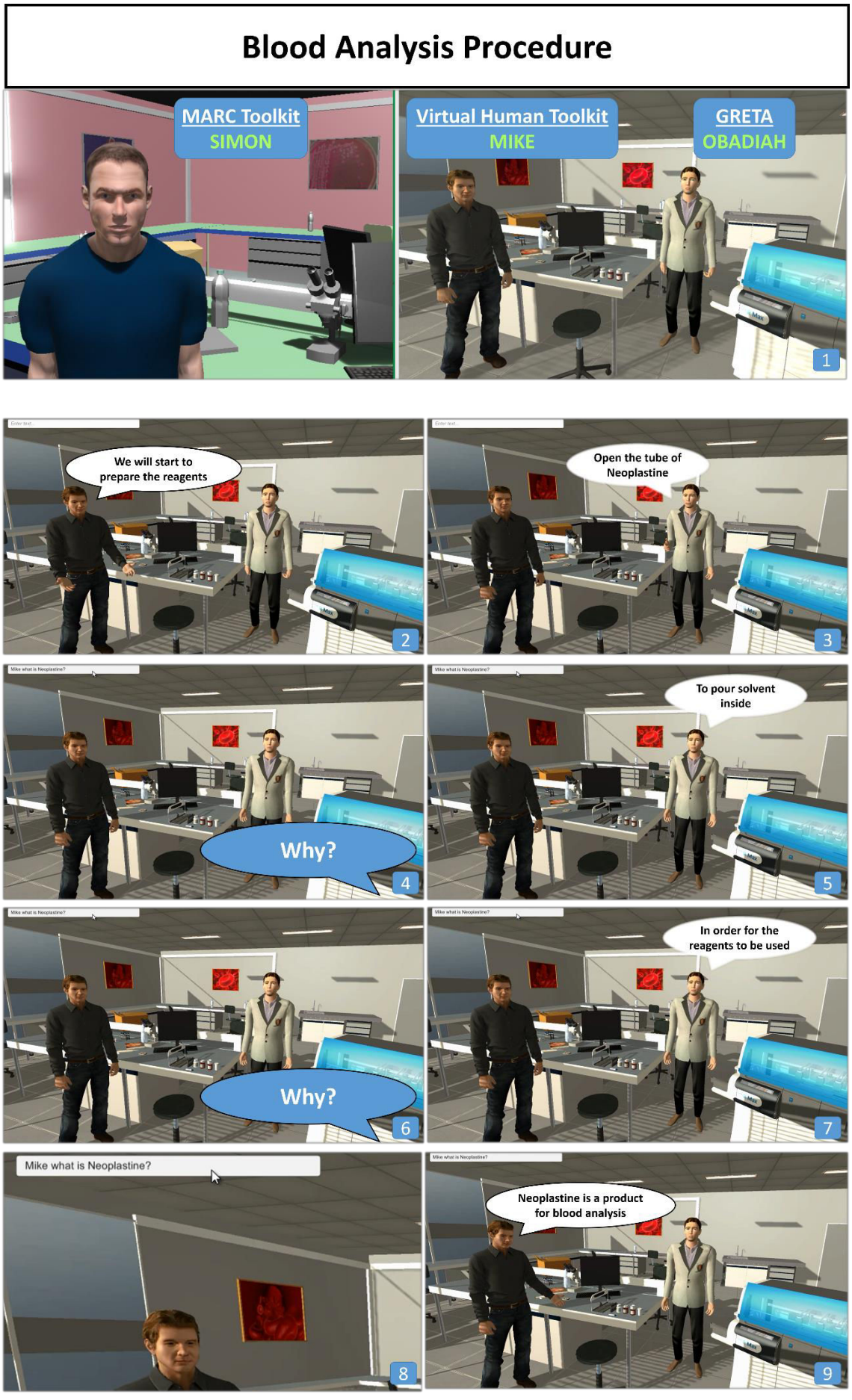
Blood Analysis Procedure
MARC Toolkit
SIMON
Virtual Human Toolkit
MIKE
GRETA
OBADIAH
1
We will start to prepare the reagents
2
Open the tube of Neoplastine
3
Why?
4
To pour solvent inside
5
Why?
6
In order for the reagents to be used
7
Mike what is Neoplastine?
8
Neoplastine is a product for blood analysis
9

Figure 63. The implemented scenario of the blood analysis procedure

5 EVALUATION

Similar to previous evaluations [34], we want to validate the impact of our model on the natural interaction between a user and an ECA, more precisely in a VLE, having the possibility to exchange knowledge by communications using the model we proposed in Chapter 3. In order to do this, we choose the context of our application, which is a virtual environment to learn a procedure (Chapter 4).

5.1 Experiment protocol

In the experiment we settled, the learner has to follow the blood analysis procedure which implies the manipulation of virtual objects in the virtual environment. Initially, the learner has to retrieve the information about the procedural actions before executing them. She/he can also consult the instructions about the execution of an action.

We use the same protocol as Hoareau et al [35] then, the experiment requires the learner to repeat the procedure for several trials. The learner could acquire the procedural knowledge to her/his memory. We make the hypothesis that upon repetition, the performance of learning will be better. In Hoareau, the performance is represented by the time of execution, the number of errors and the number of assistances.

The hypothesis of our experiment is to assume that the presence of an Embodied Conversational Agent (ECA) in a Virtual Environment (VE) enhances the learning performance (or at least does not degrade it) in the context of a learning procedure.

The procedure of this experiment is composed of 38 actions (see Appendix 3) that the learner should realize for several trials.

We define two different tutoring conditions (Condition 1: with ECA, Condition 2: without ECA).

Forty participants with different ages and academic backgrounds from the academic institute, Arts, Sciences and Technology University in Lebanon (AUL), were involved in this study. Each participant played the role of the learner whose goal is to perform the procedure for seven trials. They were unaware of the procedure and how many rounds (trials) the experiment takes. We separated them in two groups of twenty participants with different experiment conditions (Group 1 – 20 participants - Condition 1, Group 2 - 20 participants - Condition 2).

5.1.1 Description of the experiment

In this experiment, each learner has to execute the procedure in seven consecutive trials. Several measures are gathered in every trial in order to be studied and to observe the variations in the activities and the collected results. The consumed time to complete the procedure, the number of consultations of the instructions, and the number of committed errors, are the measures that we considered for the evaluation. Subsequently, a log file with these measures is generated after every trial of each participant.

Figure 64. Interactive menu for selecting actions

The scenario of the experiment is executed in a virtual environment that represents the blood analysis laboratory (Figure 65). The learner is requested to perform the procedure of the blood analysis tests on the automaton biological analysis machine. To perform

this procedure, the learner must follow up the given instructions in order to select suitable actions and manipulate the right objects. To manipulate an object and perform the required action, the learner has to press the left mouse button over an object, use the "up and down" keyboard arrow keys to select an action, and then to press the "Enter" button in order to confirm her/his selection (Figure 64).

5.1.1.1 Experiment with HELP icon

In the scenario of the experiment with the first condition (Condition 1), the instructions to perform the actions of the procedure are vocally given. The learner can press the question-mark icon (), which is displayed at the bottom right corner of the screen (Figure 65), in order for vocal instruction to be given or repeated. After pressing this icon, a synthetic voice provides the learner with the description of the next action to perform, and the object that should be manipulated blinks red until the learner presses on it and selects an action (Figure 65). The learner can repeatedly refer to these instructions as much as needed.

Figure 65. Experiment with HELP icon

5.1.1.2 Experiment with an ECA

The scenario with the second condition (Condition 2) shows the integration of an ECA-tutor in the virtual environment (Figure 66). The learner has to vocally communicate with the ECA-tutor to inquire about the next action by clearly saying "NEXT". The

ECA responds and points to the object that should be manipulated while it is blinking in red (Figure 66). The learner can repeatedly refer to the instructions as much as needed.

Figure 66. Experiment with an ECA

5.1.2 Log files

To automatically get the information on the evolution of the performance of learning of the user, we developed our system to generate a log file, after each trial of every participant in the experiment, to gather the following objective performance measures:

- Total time of execution: the start and end date time of the trial
- Number and date of requested assistance: the number and the date of consultations for instructions
- Realized action info: realized date, action name, manipulated object, and the validity of the performed action (wrong or right)

5.1.3 Questionnaire

Upon finishing the seventh trial in the experimental session, each learner should apply the multi-dimensional subjective workload technique using the questionnaire of the NASA Task Load Index (TLX) [101] [102]. Figure 67 shows the rating sheet that includes several rating-scale dimensions (Mental Demand, Physical Demand, Temporal Demand, Performance, Effort and Frustration Level) which the learner has to scale. The NASA TLX questionnaire uses bipolar scales to rate these rating-scale dimensions

(Figure 67). The learner uses this questionnaire to evaluate the experiment they underwent. The second part of the TLX questionnaire asks the learner to weight the importance of these rating-scales by selecting the scales that could primarily affect their performance for executing the procedure.

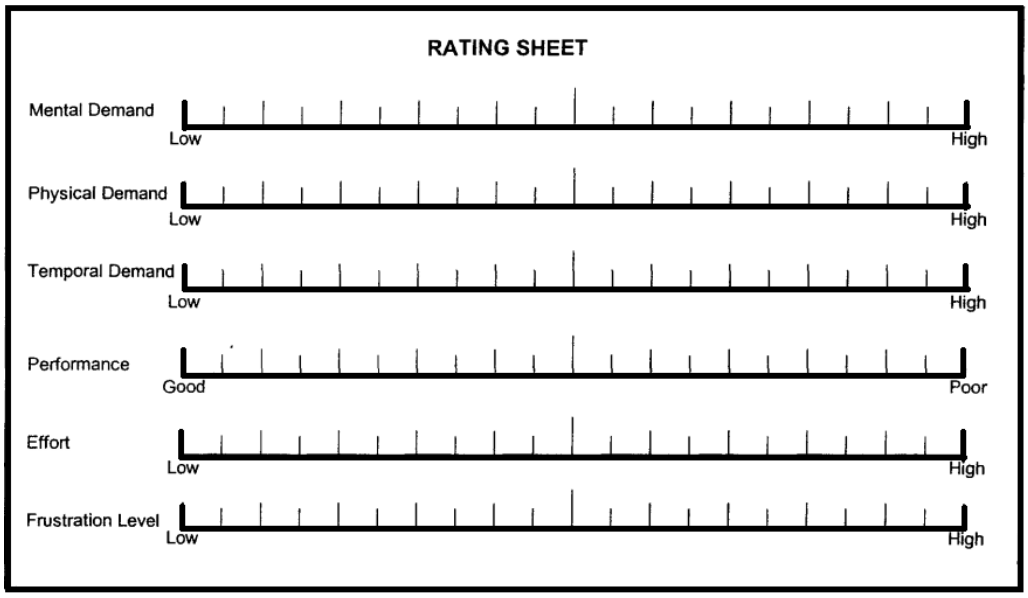

Figure 67. NASA TLX questionnaire rating scale

5.2 Results

To evaluate the experiment and validate the impact of the proposed facts in our model, we execute the formulas of several statistical tests on the collected performance measures in the log files.

While studying these log files, we recognized that the three last trials correspond to the third phase of learning (the autonomous phase) [103]. In this phase, the learner is just trying to improve her/his performance and not getting new knowledge. Accordingly, we consider that we have just to compare the results of the two first phases (Cognitive phase and Associative phase) using the four first trials where our proposition can have an impact.

The objective performance measures are represented in the statistical formulas as Dependent Variables (DV). We consider the funnel concept to organize the adopted performance measures (1- Execution time → 2- Consulting assistance → 3- Number of errors) (Figure 68). A funnel is used to represent the measures of the experiment, since a prospect descends into the next stage of the funnel when its interest increases.

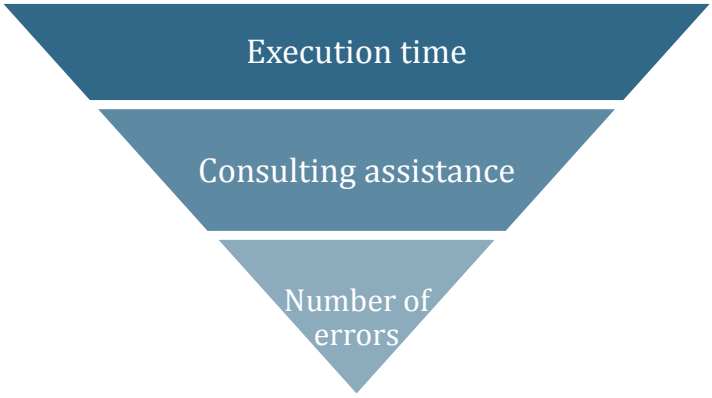

Figure 68. The funnel design concept of the objective performance measures

Besides, we study the Independent Variables (IV) which are the different conditions considered in the experiment. It might affect the objective performance measures. In our experiment, we have two IVs:

1- Presence or absence of a virtual agent (ECA)
2- Number of trials

To prove or deny the proposed hypothesis (the presence of an ECA in a VE enhances learning performance or at least does not degrade it in the context of a learning procedure), we developed a script using the "R"[11] software (Copyright to The R Foundation for Statistical Computing) to execute reliable statistical tests to evaluate the results of the experiment.

In the following sections, we present the effect of each DV (Execution time, Consulting assistance and Number of errors) on the performance of the learner among the applied tests. The results of these tests are determined by considering the probability values (p-value), and the alpha significance level (x) that is frequently set to (0.05).

5.2.1.1 Execution Time

The condition of this application is to know if we do parametric or non-parametric statistical tests. In these two parametric tests, we have to get $p > x$ in order to be significant:

[11] https://www.r-project.org/ (accessed November, 2017)

DV: Time	
Shapiro-Wilk Normality Test	
Data	Time (execution time)
p.value	0.0142097
Result	p < x → Not significant, so we do not respect the normality
F test of homogeneity of variances (F Fisher Test)	
Data	Time by Condition
p.value	0.02261723
Result	p < x → Not Significant, so we don't respect the homogeneity of variances

Since we do not have significant results in these two tests, we will do non-parametric statistical tests. We need to check for the effect of number of trials on Time (IV corresponds to the number of trials). Here, we have to get p < x in order to be significant:

Friedman rank sum test (for all participants). We use this test when we have more than two trials. The data of each trial is listed in Table V (the mean and the standard deviation of the results of all participants in every trial), and graphically represented in Figure 69.

DV: Time	
Friedman: Effect of Number of Trials / Time	
Data	Responses
p.value	2.2e-16
Result	p < x → Significant

Table V. Friedman: Effect of Number of Trials / Time in minutes

Trial	1	2	3	4
Mean	5.076766	4.429500	3.121051	2.835630
Standard Deviation	1.897760	2.032023	1.863305	1.242661

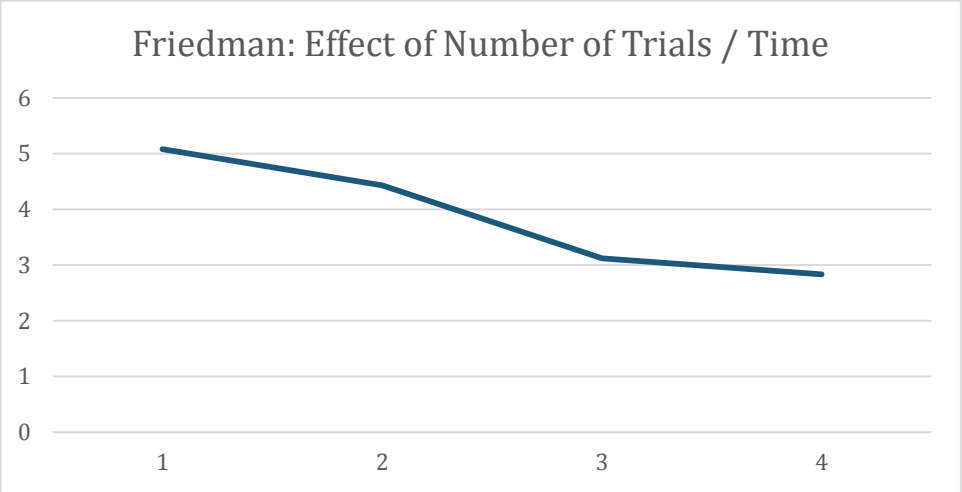

Figure 69. Friedman: Effect of Number of Trials / Time in minutes

We can reason on the obtained results of Friedman's test to say that the number of trials has an effect on the time needed to execute a trial. The time decreases significantly according to the executed trials. This is a classical result in the learning curve. It means that using our system, the learner is learning.

We also need to check for the effect of the presence of the virtual agent on the time (IV corresponds to the presence or the absence of the virtual agent):

1. Wilcoxon rank sum test with continuity correction (two independent groups). The data listed in Table VI (the mean and the standard deviation of the results of all participants in both conditions separately) and graphically represented in Figure 70 is used in this test:

DV: Time	
Wilcoxon: Effect of Condition / Time	
Data	Time by Condition
IV corresponds to the presence or absence of the virtual agent	
p.value	0.000166
Result	p < x → Significant, so the presence of the virtual agent has an effect on the time, and the time varies according to the presence or the absence of the virtual agent

Table VI. Wilcoxon: Effect of Condition / Time in minutes

	Mean	Standard Deviation
Condition-ECA	4.548177	2.133259
Condition-Icon	3.183296	1.579927

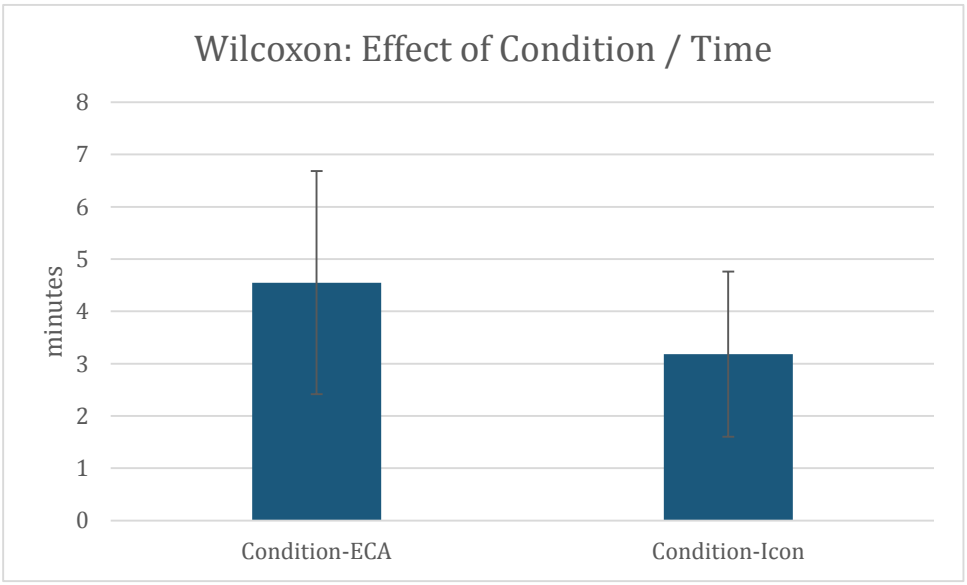

Figure 70. Wilcoxon: Effect of Condition / Time in minutes

We can reason on the obtained results of Wilcoxon's test to say that the presence of the virtual agent has an effect on the time. Interacting with a virtual agent is consuming more time than just clicking on an icon to enquire about the desired action. It means that using a virtual agent in our system need additional time from the learner.

We will check now for the effect of the presence or the absence of the virtual agent at each trial on the time consumed to finish the procedure. The data listed in Table VII (the average time consumed in every trial by all participants in both conditions separately) and graphically represented in Figure 71 is used in these tests:

DV: Time	
Trial 1: Wilcoxon: Effect of Condition / Time for Trial1	
Data	Time by Condition
p.value	0.04084
Result	p < x → Significant

We notice that there is a time difference between the two conditions in the first trial, so we have to refer to the average table (Table VII) to check which condition took more time to finish the procedure.

DV: Time	
Trial 2: Wilcoxon: Effect of Condition / Time for Trial2	
Data	Time by Condition
p.value	0. 001963
Result	p < x → Significant
Trial 3: Wilcoxon: Effect of Condition / Time for Trial3	
Data	Time by Condition
p.value	0. 003155
Result	p < x → Significant

Trial 4: Wilcoxon: Effect of Condition / Time for Trial4	
Data	Time by Condition
p.value	0. 002702
Result	p < x → Significant

Table VII. Wilcoxon: Effect of Condition / Time for each Trial

Trials	1	2	3	4
Condition-ECA	5.463996	5.340858	3.869896	3.517959
Condition-Icon	4.689536	3.518142	2.372206	2.153302

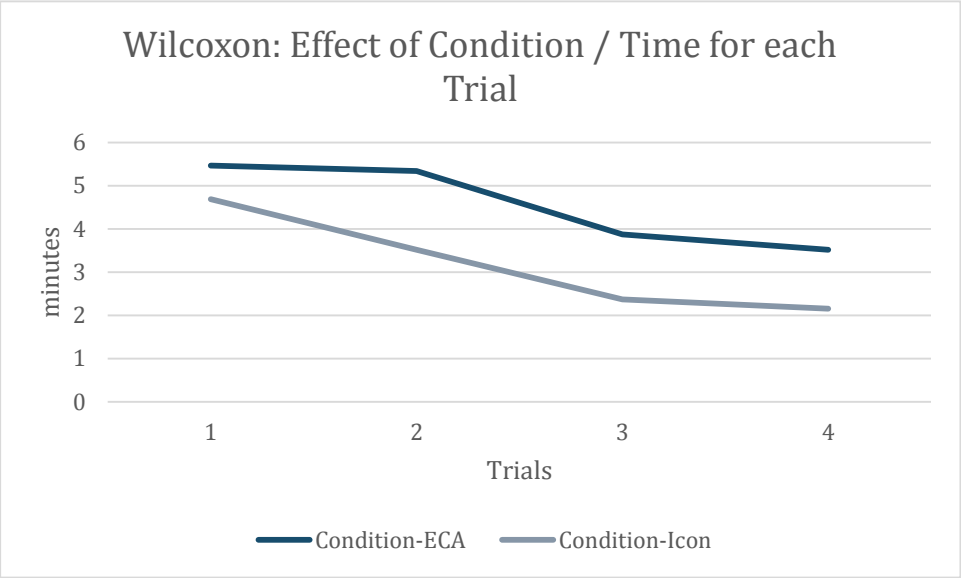

Figure 71. Wilcoxon: Effect of Condition / Time for each Trial

We can reason on the obtained results of Wilcoxon's test, by applying the trials of the experiment on the two conditions, to say that the number of trials has an effect on the time while applying the two conditions (Condition-ECA and Condition-Icon). The time decreases significantly according to the executed trials. This is a classical result in the learning curve. It means that using our system in any of the two conditions, the learner is learning.

We can notice that the presence of the virtual agent has an effect on the time consumed by the participants (in Condition-ECA) during the performed trials of the experiment.

5.2.1.2 Consulting assistance (Help)

The condition of this application is to know if we do parametric or non-parametric statistical tests. In these parametric tests, we have to get p > x in order to be significant:

DV: number of times asking for help (number of times to consult instructions)	
Shapiro-Wilk Normality Test	
Data	Help
p.value	2.884966e-10
Result	p < x → Not significant, so we do not respect the normality
F test of homogeneity of variances (F Fisher Test)	
Data	Help by Condition
p.value	0.4908131
Result	p > x → Significant, so we respect the homogeneity of variances

Since not both tests are significant, we have to do non-parametric statistical tests. We have to check for the effect of number of trials on number of Help consultations (IV corresponds to the number of trials). Here, we have to get p < x in order to be significant.

1. Friedman rank sum test. The data listed in Table VIII (the mean and the standard deviations of the results of all participants in every trial) and graphically represented in Figure 72 is used in this test:

DV: Help	
Friedman: Effect of Number of Trials / Help	
Data	Responses
IV corresponds to the number of trials	
p.value	0.0003501
Result	p < x → Significant

Table VIII. Friedman: Effect of Number of Trials / Help

Trial	1	2	3	4
Mean	36.166667	18.6	6.833333	3.866667
Standard Deviation	9.667162	11.254424	9.843897	7.532611

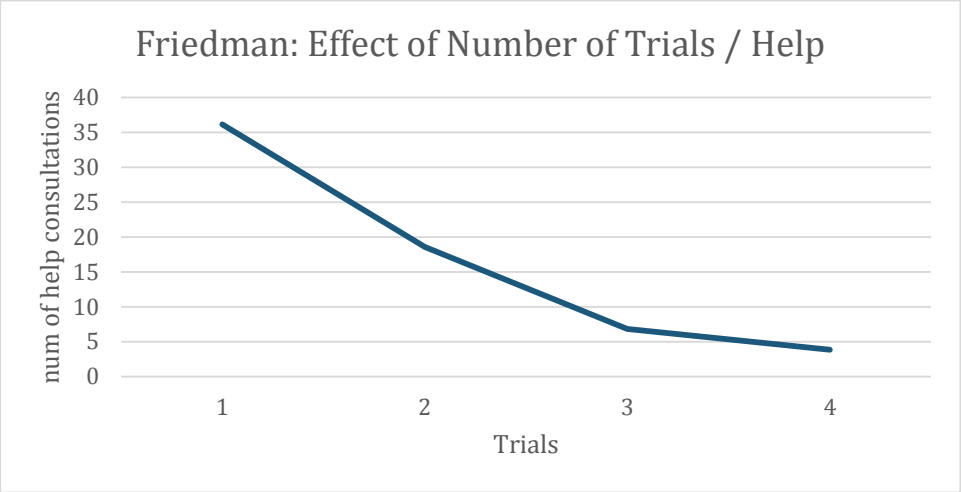

Figure 72. Friedman: Effect of Number of Trials / Help

We can reason on the obtained results of Friedman's test to say that the number of trials has an effect on the help consultations. The number of times asking for help decreases significantly according to the executed trials. This means that using our system, the learner is learning.

We present here the effect of the presence of the virtual agent on the number of Help consultations (IV corresponds to the presence or absence of the virtual agent).

2. Wilcoxon rank sum test with continuity correction. The data listed in Table IX (the mean and the standard deviation of the results of all participants in both conditions separately) and graphically represented in Figure 73 is used in this test:

DV: Help	
Wilcoxon: Effect of Condition / Help	
Data	Help by Condition
IV corresponds to the presence or absence of the virtual agent	

p.value	0.2295
Result	p > x → Not Significant

Table IX. Wilcoxon: Effect of Condition / Help

	Mean	Standard Deviation
Condition-ECA	18.26667	16.57123
Condition-Icon	14.46667	15.14279

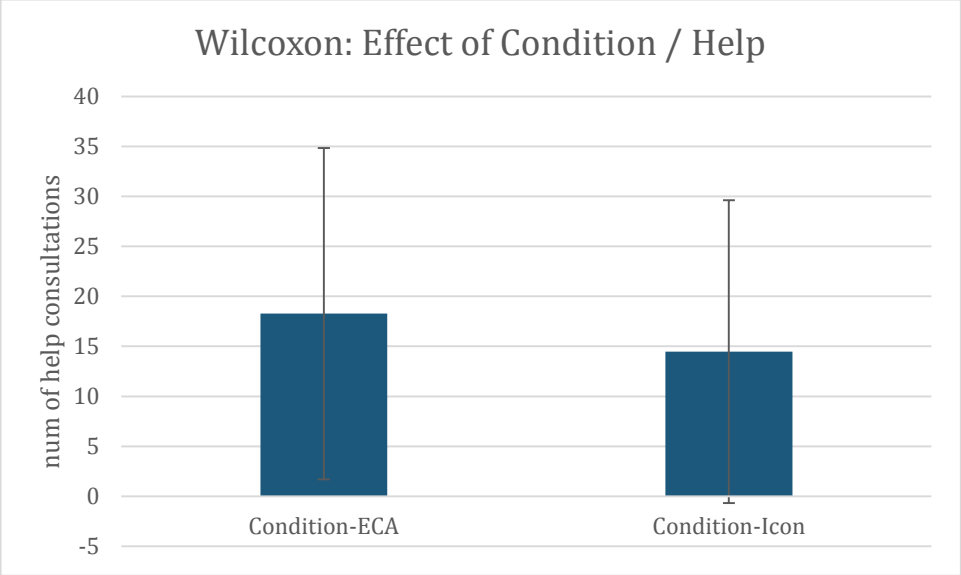

Figure 73. Wilcoxon: Effect of Condition / Help

We are going to check for the effect of the presence or the absence of the virtual agent at each trial on the number of help consultations. The data listed in Table X (the average of the number of help consultations in every trial by all participants in both conditions separately) and graphically represented in Figure 74 is used in these tests:

DV: Help	
Trial 1: Wilcoxon: Effect of Condition / Help for Trial1	
Data	Help
p.value	0.2026

Result	p > x → Not Significant
Trial 2: Wilcoxon: Effect of Condition / Help for Trial2	
Data	Help
p.value	0.2897
Result	p > x → Not Significant
Trial 3: Wilcoxon: Effect of Condition / Help for Trial3	
Data	Help
p.value	0.4012
Result	p > x → Not Significant
Trial 4: Wilcoxon: Effect of Condition / Help for Trial4	
Data	Help
p.value	0.405
Result	p > x → Not Significant

Table X. Wilcoxon: Effect of Condition / Help for each Trial

Trials	1	2	3	4
Condition-ECA	37	21.26667	9	5.8
Condition-Icon	35.33333	15.93333	4.666667	1.933333

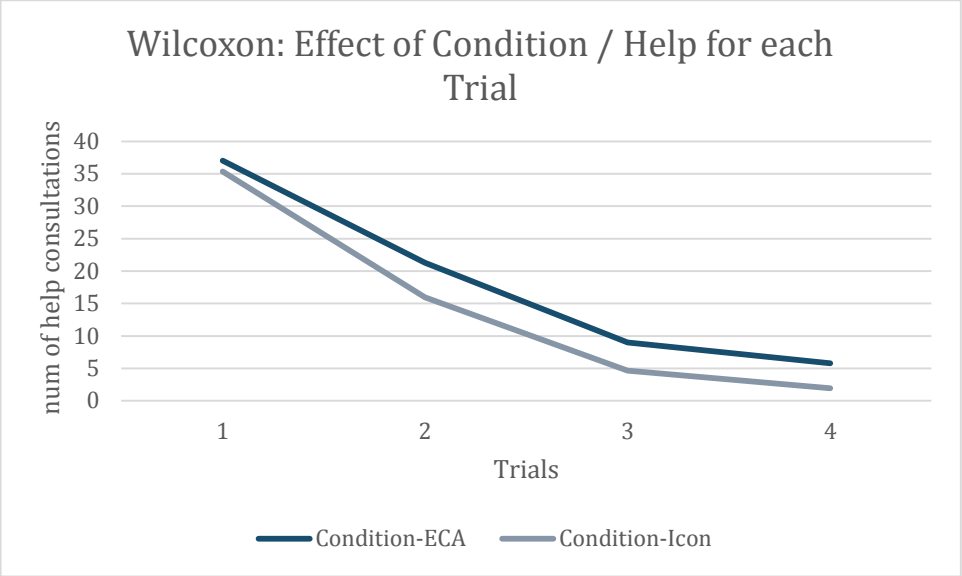

Figure 74. Wilcoxon: Effect of Condition / Help for each Trial

According to the above results, we can notice that we do not have a significant difference in the number of help consultations inquired by the participants in both conditions during the performed trials of the experiment. But according to the obtained results of Wilcoxon's test, we can say that the number of trials has an effect on the number of help consultations in both conditions. The number of times asking for help decreases significantly according to the executed trials. It means that using our system, the learner is learning.

5.2.1.3 Number of errors (Incorrect Actions)

The condition of this application is to know if we do parametric or non-parametric statistical tests. In these two parametric tests, we have to get $p > x$ in order to be significant:

DV: Number of incorrect actions	
Shapiro-Wilk Normality Test	
Data	Errors (number of errors in actions)
p.value	3.015709e-15
Result	$p < x \rightarrow$ Not significant, so we do not respect the normality
F test of homogeneity of variances (F Fisher Test)	
Data	Errors by Condition

p.value	8.482104e-13
Result	p < x → Not Significant, so we do not respect the homogeneity of variances

Since we don't have significant results in these two tests, we will do non-parametric statistical tests. We have to check for the effect of number of trials on number of incorrect actions (IV corresponds to the number of trials). Here, we have to get p < x in order to be significant.

1. Friedman rank sum test (for all participants). We use this test when we have more than two trials. The data listed in Table XI (the mean and the standard deviations of the results of all participants in every trial) and graphically represented in Figure 75 is used in this test:

DV: Number of incorrect actions	
Friedman: Effect of Number of Trials / Incorrect Actions	
Data	Responses
IV corresponds to the number of trials	
p.value	1.125e-05
Result	p < x → Significant

Table XI. Friedman: Effect of Number of Trials / Incorrect Actions

Trial	1	2	3	4
Mean	2.133333	8.933333	8.566667	5.766667
Standard Deviation	2.270247	11.682387	8.071591	5.437313

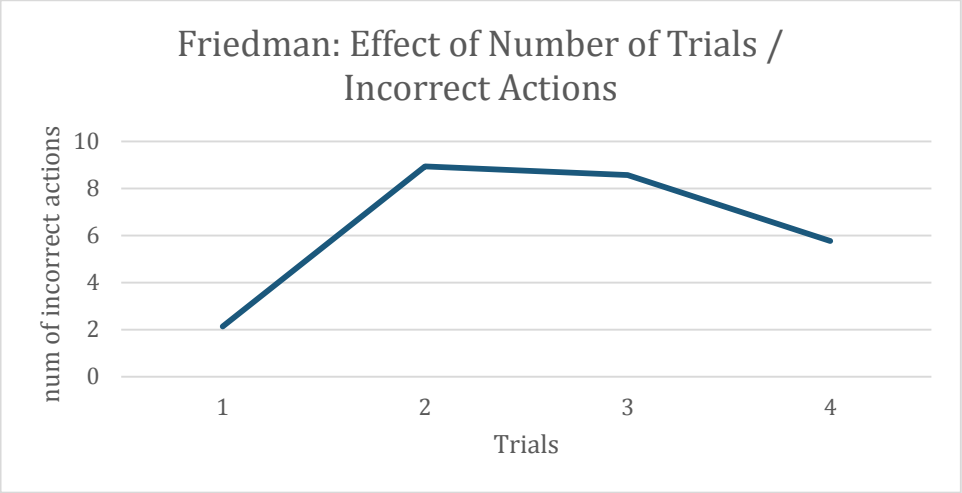

Figure 75. Friedman: Effect of Number of Trials / Incorrect Actions

We can reason on the obtained results of Friedman's test to say that the number of trials has an effect on the number of incorrect actions. After executing the first trial to recognize the required actions, the number of times for making fault actions decreases significantly according to the executed trials. This means that using our system, the learner is learning.

We have to acquire the effect of the presence of the virtual agent on the number of incorrect actions (IV corresponds to the presence or absence of the virtual agent).

2. Wilcoxon rank sum test with continuity correction. The data listed in Table XII (the mean and the standard deviation of the results of all participants in both conditions separately) and graphically represented in Figure 76 is used in this test:

DV: Number of incorrect actions	
Wilcoxon: Effect of Condition / Incorrect Actions	
Data	Errors by Condition
IV corresponds to the presence or absence of the virtual agent	
p.value	0.3053
Result	p > x → Not Significant, so the presence of the virtual agent has no effect on the number of errors

Table XII. Wilcoxon: Effect of Condition / Incorrect Actions

	Mean	Standard Deviation
Condition-ECA	7.933333	10.547498
Condition-Icon	4.766667	3.863636

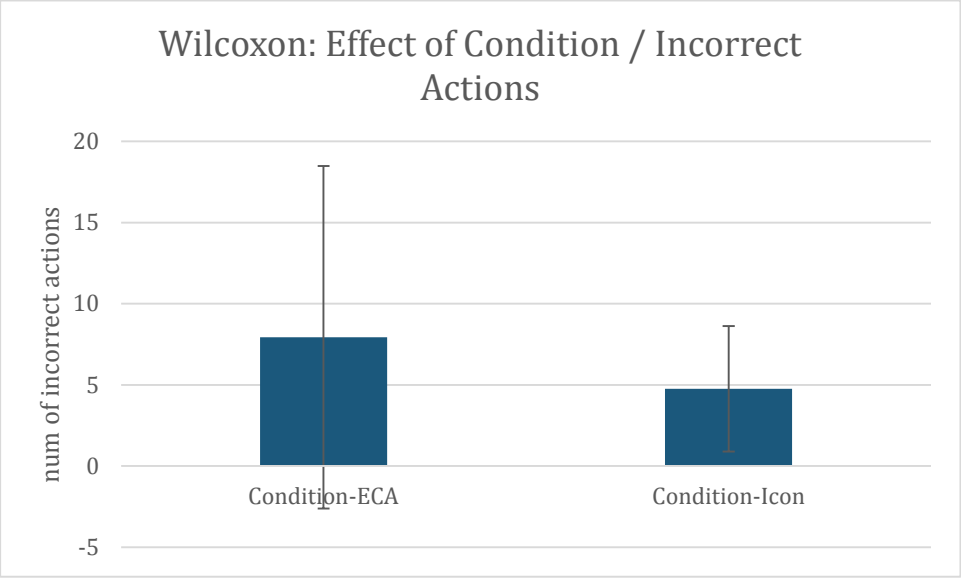

Figure 76. Wilcoxon: Effect of Condition / Incorrect Actions

We can reason on the obtained results of Wilcoxon test to say that applying the different conditions has no effect on the number of incorrect actions. The number of times for making fault actions decreases significantly according to the executed trials. This means that using our system, the learner is learning in both conditions.

We will check now for the effect of the presence or the absence of the virtual agent at each trial on the number of incorrect actions. The data listed in Table XIII (the average of the number of incorrect actions in every trial by all participants in both conditions separately) and graphically represented in Figure 77 is used in these tests:

DV: Number of incorrect actions	
Trial 1: Wilcoxon: Effect of Condition / Incorrect Actions for Trial1	
Data	Errors
p.value	1
Result	p > x → Not Significant

Trial 2: Wilcoxon: Effect of Condition / Incorrect Actions for Trial2	
Data	Errors
p.value	0.9336
Result	p > x → Not Significant
Trial 3: Wilcoxon: Effect of Condition / Incorrect Actions for Trial3	
Data	Errors
p.value	0.1968
Result	p > x → Not Significant
Trial 4: Wilcoxon: Effect of Condition / Incorrect Actions for Trial4	
Data	Errors
p.value	0.1748
Result	p > x → Not Significant

Table XIII. Wilcoxon: Effect of Condition / Incorrect Actions for each Trial

Trials	1	2	3	4
Condition-ECA	2	10.933333	11.2	7.6
Condition-Icon	2.266667	6.933333	5.933333	3.933333

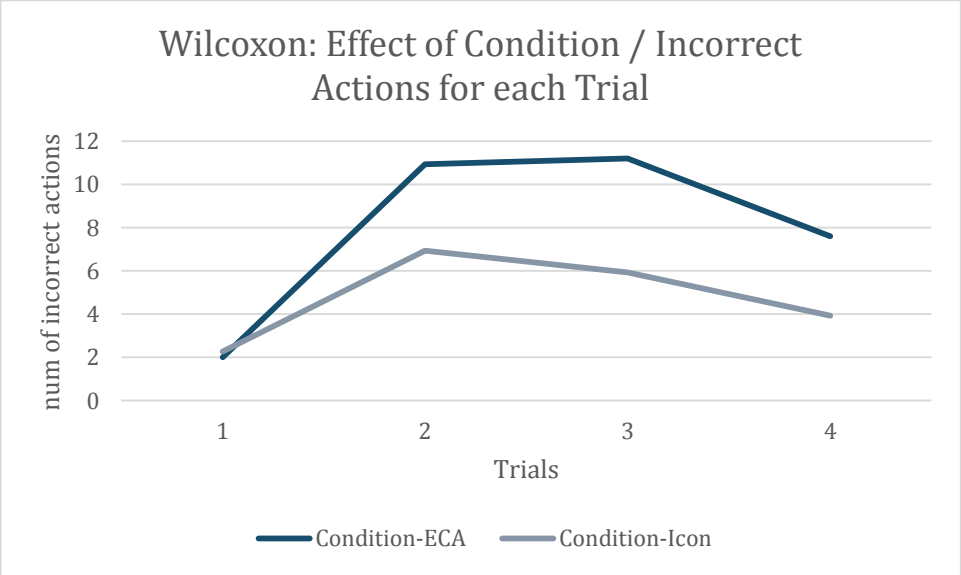

Figure 77. Wilcoxon: Effect of Condition / Incorrect Actions for each Trial

According to the above results, we can notice that we do not have a significant difference in the number of incorrect actions committed by the participants in the two conditions among all the trials of the experiment. But according to the obtained results of Wilcoxon's test, we can say that the number of trials has an effect on decreasing the number of incorrect actions in both conditions. The number of incorrect actions decreases significantly according to the executed trials. It means that using our system, the learner is learning.

Discussion

As we previously declared, the purpose of evaluating the experiment is to find the effect of the existence of the Independent Variables (Presence or absence of a virtual agent, and Number of trials) on the Dependent Variables (Execution time, Consulting assistance, and Number of errors) based on the results of the statistical tests in order to be able to confirm or deny the proposed hypothesis (we assume that the presence of ECA in a VE enhances the learning performance in the context of a learning procedure). By referring to the statistical test results, we conclude that there is no significant effect for the presence or the absence of the ECA on the "Help" and "Incorrect Actions" DVs. We can also notice that it takes more time for participants to perform the procedure in each trial when they have to vocally communicate with the ECA rather than just pressing on the "?" icon. However, what might be important is the difference between the measures of trials and not their values. The difference between the measures of trials when having an ECA is greater than that when using the "?" ICON (Table V, Table VIII and Table XI). Accordingly, we can say that a better learning performance could be gained when an ECA is used.

To confirm the obtained results, we conducted the same experiment on an expert of the procedural scenario of the blood analysis. The same results were attained.

As a conclusion, we use these results to partially confirm the hypothesis of the applied experiment and assure that having an ECA in the virtual environment does not degrade the performance of the learner in the context of a learning procedure. More effects of IVs can probably be studied when executing an advanced procedure with a better pedagogical behavior and by including more aspects in studying the performance of the learner.

6 CONCLUSION & PERSPECTIVES

In this thesis, we proposed a novel model for conceiving and implementing VLE with intelligent virtual agents having a BDI-like based cognitive architecture and materialized through ECAs that ensure human-like and credible interactions with the user during the progress of the pedagogical scenarios. The proposed cognitive architecture represents both, the knowledge on the environment and the internal state of the agents and their evolution. It allows to plugin rich reasoning models to draw out sound decisions and intentions.

Our model is implemented on top of MASCARET. This allows domain experts, with no or little technical background, to define their pedagogical scenarios using UML. They can also reuse artifacts from previous scenarios and easily extend existing ones. We extend MASCARET with bricks to implement our agents' cognitive architecture, and we use SAIBA to map agents' behaviors to concrete ECA interactions.

Using our implemented model, we built a biomedical pedagogical scenario, in which virtual tutors guide laboratory workers to learn and apply a blood analysis procedure. The learner is advised to follow the default sequence of actions needed to successfully accomplish the procedure, and receives the right directions even when unintentional disruptions interrupts the normal sequel of the procedure.

We used this tutoring system to apply an experimental scenario on several participants. We applied the experimental protocols defined by Hoareau et al. [35] to evaluate the

experiment and to validate the impact of our model based on objective performance measures. We used the results of this evaluation to partially confirm the hypothesis of the experiment and assure that having an ECA in the VLE does not degrade the performance of the learner in the context of a learning procedure.

6.1 Perspectives

We aim to implement our model on different scenarios in order to confirm the authenticity of our model. Nevertheless, we are going to indicate in this section the several perspectives that can extend our work to include additional functionalities for the VLE.

6.1.1 Building an advanced Tutoring Behavior

The agent in our model is able to answer to questions, but it can be interesting if this agent can ask questions to the learner while executing the domain procedure and after finishing it in order to make sure that the domain procedure is well tutored. This can be acquired by checking the learner's knowledge. An examination pedagogical scenario can be defined and linked to the domain procedure. While executing the domain procedure, the tutor agent can perform pedagogical actions in this scenario to ask the learner about acquired information to make sure that she/he can succeed in continuing the procedure.

Consequently, going further on interpretation of the natural language of the learner is a mandatory operation that has to be implemented in order to parse the answers of the learner. The generic AIML patterns, which are defined to interpret the learner's dialogue, should be developed to understand the domain terms and the answers of the learner.

The learner could not realize how to perform a certain action in the domain procedure. Therefore, the tutor agent should be able to execute all domain actions, and the learner can then request from a tutor agent to perform the difficult actions. Necessary performative functions and slots have to be used in the COMMUNICATIONACTIONS that are exchanged between the agents to implement the action-requests of the learner.

To consider more human interaction behaviors in the VLE, the facial expressions of the learner could be detected in order to recognize her/his feedback after being informed about performing a domain action. For example, when the tutor agent asks the learner to manipulate an object in the environment, if the learner expresses a negative feedback,

the agent can provide additional information about that entity, like its role or any other property.

In a similar way, facial expressions could be as well realized by the tutor agents to express natural feedback in the virtual environment. The tutor agent has to analyse all actions performed by the learner and realize necessary facial expressions. For example, when the learner performs for several times the same incorrect action, in addition for informing the learner with the correct notes, the tutor agent can realize a wondering facial expression to naturally react to the faulty performance of the learner.

To implement these perspectives, further work on the ECA and on an Intelligent Tutoring System (ITS) in the domain application of the VLE have to be implemented.

6.1.2 Intelligent Tutoring System

The intelligent tutoring system (ITS) is a computer system that can be used to tutor the user on the knowledge of a certain domain through communication and interaction. The major common objective of all ITSs is to support the learning processes of handled scenarios by supplying it with appropriate tutoring services.

The architecture of the ITS consists of four models (Domain model, Student model, Tutoring model and Interface model) [104] (Figure 78) which require studying and analyzing the knowledge patterns and the reasoning sources [105] in the VLE. This analysis can specify the behaviors of the agents to properly interact with the learner and the environment.

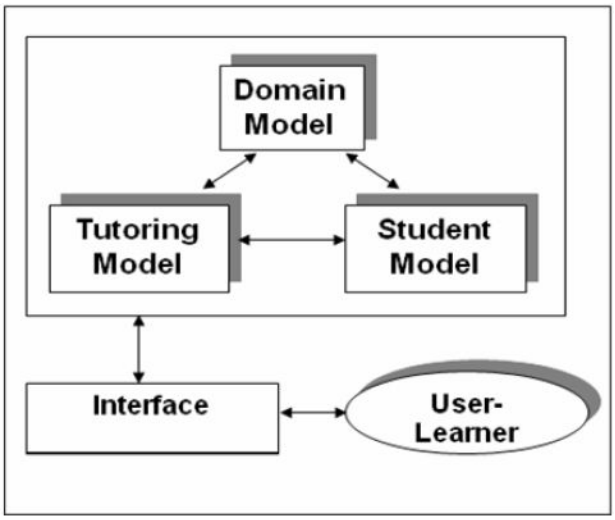

Figure 78. The four-component architecture of an ITS

The domain knowledge provided by the domain experts is represented in the domain model (declarative knowledge), while the pedagogical knowledge is defined by the domain instructors to provide the reasoning facts to demonstrate the declarative knowledge [106].

While executing the learning scenario, the student model carries the knowledge about the achieved progress of the learner. When the learner interacts with the environment, this model tracks the activities, the cognitive states and the attained knowledge of the learner. These assets are considered as the reasoning aspects that are required to execute suitable actions.

The interactions between the learner and the agents are well managed in the ITS. The ITS follows pedagogical strategies that are educationally well tested. The tutoring model considers the knowledge aspects of the domain and student models in order to select the pedagogical and tutoring activities that can assist the learner. This model is also responsible for replying to the inquiries of the user [104] [107].

The interface model permits the interaction and the exchange of information between the learner and the system. Different types of interface behaviors, such as text, visual or auditory fields, are adopted to translate the system information to the learner. Besides, several input sources and communication conventions, such as microphone, keyboard, mouse and joystick can be used by the learner to interact with the system [108].

Many ITS models, such as [109], [110], [111] and [112] are previously released and used. The ITS of [109] includes a web-based environment for teaching geometry proofs. While in [110], the ITS automatically categorizes the contents of the virtual environment, but can only provide a poor interaction techniques with the learner. Whereas, [111] defines a simulated framework that acts as an ITS and includes a scenario authoring tool for the medical domain.

7 BIBLIOGRAPHY

[1] 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," *Computer Modeling in Engineering and Sciences (CMES),* vol. 47(3), p. 299, 2009.

[2] G. A. Giraldi, R. Silva and J. C. Oliveira, "Introduction to virtual reality," *LNCC Research Report,* vol. 6, 2003.

[3] T. Mazuryk and M. Gervautz, "Virtual reality-history, applications, technology and future," in *Institute of Computer Graphics, Vienna University of Technology*, Austria, 1996.

[4] S. K. Ong and A. Y. C. Nee, "Virtual and augmented reality applications in manufacturing," *Springer Science & Business Media,* 2013.

[5] E. Gobbetti and R. Scateni, "Virtual reality: Past, present, and future," *Virtual environments in clinical psychology and neuroscience: Methods and techniques in advanced patient-therapist interaction,* 1998.

[6] H. Hsiu Mei and L. Shu Sheng, "Applying situated learning in a virtual reality system to enhance learning motivation," *International Journal of Information and Education Technology,* vol. 1, p. 298, 2011.

[7] M. Sacco, G. Dal Maso, F. Milella, P. Pedrazzoli, D. Rovere and W. Terkaj,

"Virtual Factory Manager," *Virtual and Mixed Reality-Systems and Applications*. *Springer Berlin/Heidelberg*, pp. 397-406, 2011.

[8] L. Beyer-Berjot, V. Palter, T. Grantcharov and R. Aggarwal, "Advanced training in laparoscopic abdominal surgery: a systematic review," *Surgery,* vol. 156(3), pp. 676-688, 2014.

[9] A. G. Gallagher, N. E. Seymour, J. A. Jordan-Black, B. P. Bunting, K. McGlade and R. M. Satava, "Prospective, randomized assessment of transfer of training (ToT) and transfer effectiveness ratio (TER) of virtual reality simulation training for laparoscopic skill acquisition," *Annals of surgery,* vol. 257(6), pp. 1025-1031, 2013.

[10] L. Greunke and A. Sadagic, "Taking Immersive VR Leap in Training of Landing Signal Officers," *IEEE transactions on visualization and computer graphics*, vol. 22(4), pp. 1482-1491, 2016.

[11] H. C. Miles, S. R. Pop, S. J. Watt, G. P. Lawrence, N. W. John, V. Perrot, P. Mallet and D. R. Mestre, "Investigation of a virtual environment for rugby skills training," *In Cyberworlds (CW), 2013 International Conference*. *IEEE,* pp. 56-63, 2013, October.

[12] S. Gerbaud, N. Molle, F. Ganier, B. Arnaldi and J. Tisseau, "GVT: a platform to create virtual environments for procedural training," *Virtual Reality Conference, 2008*. *VR'08*. *IEEE,* pp. 225-232, 2008.

[13] P. Dillenbourg, D. Schneider and P. Synteta, "Virtual learning environments," *3rd Hellenic Conference "Information & Communication Technologies in Education"*, pp. 3-18, 2002.

[14] T. Huber, M. Paschold, C. Hansen, T. Wunderling, H. Lang and W. Kneist, "New dimensions in surgical training: immersive virtual reality laparoscopic simulation exhilarates surgical staff," *Surgical Endoscopy, 1-6,* 2017.

[15] V. N. Palter and T. P. Grantcharov, "Individualized deliberate practice on a virtual reality simulator improves technical performance of surgical novices in the operating room: a randomized controlled trial," *Annals of surgery,* vol. 259(3), pp. 443-448, 2014.

[16] R. Webster, "Declarative knowledge acquisition in immersive virtual learning environments," *Interactive Learning Environments,* vol. 24(6), pp. 1319-1333, 2016.

[17] L. Edward, D. Lourdeaux, J. P. A. Barthès, D. Lenne and J. M. Burkhardt, "Modelling autonomous virtual agent behaviours in a virtual environment for risk," *IJVR,* vol. 7(3), pp. 13-22, 2008.

[18] B. Koper, R.O. and A. T, "Ims learning design information model," *IMS Global Learning Consortium,* 2003.

[19] R. Querrec, C. Buche, F. Lecorre and F. Harrouet, "Agent metamodel for virtual reality applications," *Emerging Intelligent Technologies in Industry*. *Springer Berlin Heidelberg,* pp. 81-90, 2011.

[20] R. Bouville, V. Gouranton, T. Boggini, F. Nouviale and B. Arnaldi, "#FIVE: High-Level Components for Developing Collaborative and Interactive Virtual Environments," *In Proceedings of Eighth Workshop on Software Engineering and Architectures for Realtime Interactive Systems (SEARIS 2015), conjunction with IEEE Virtual Reality (VR),* 2015, March.

[21] C. Barot, D. Lourdeaux and D. Lenne, "Using planning to predict and influence autonomous agents behaviour in a virtual environment for training," *In Cognitive Informatics & Cognitive Computing (ICCI* CC), 12th IEEE International Conference*, pp. 274-281, 2013, July.

[22] B. C. C. Souza, R. F. D. A. Bolzan, J. G. Martins and A. M. Rodriguez, "Cognitive strategies for virtual learning environments," 2000.

[23] J. Rickel and W. L. Johnson, "Steve: An animated pedagogical agent for procedural training in virtual environments," *Intelligent virtual agents, Proceedings of Animated Interface Agents: Making Them Intelligent,* pp. 71-76., 1997.

[24] J. Klein, Y. Moon and R. W. Picard, "This computer responds to user frustration: Theory, design, and results," *Interacting with computers,* vol. 14(2), pp. 119-140, 2002.

[25] J. Cassell, Embodied conversational agents, MIT press, 2000.

[26] J. C. Lester, S. A. Converse, S. E. Kahler, S. T. Barlow, B. A. Stone and R. S. Bhogal, "The persona effect: affective impact of animated pedagogical agents," *In Proceedings of the ACM SIGCHI Conference on Human factors in computing systems, ACM,* pp. 359-366, 1997.

[27] B. Friedman, Human values and the design of computer technology, Cambridge University Press, 1997.

[28] A. Nijholt, "Humor and embodied conversational agents," CTIT, 2003.

[29] H. C. Lane, M. J. Hays, M. G. Core and D. Auerbach, "Learning intercultural communication skills with virtual humans: Feedback and fidelity," *Journal of Educational Psychology 105.4,* pp. 10-26, 2013.

[30] W. L. Johnson, J. W. Rickel and J. C. Lester, "Animated pedagogical agents: Face-to-face interaction in interactive learning environments," *International Journal of Artificial intelligence in education,* vol. 11(1), pp. 47-78, 2000.

[31] J. C. Lester, J. L. Voerman, S. G. Towns and C. B. Callaway, "Deictic believability: Coordinated gesture, locomotion, and speech in lifelike pedagogical agents," *Applied Artificial Intelligence,* Vols. 13(4-5), pp. 383-414, 1999.

[32] P. Doyle, "Believability through context using knowledge in the world to create intelligent characters," *In Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1, ACM,* pp. 342-349, 2002.

[33] P. Chevaillier, T. H. Trinh, M. Barange, P. De Loor, F. Devillers, J. Soler and R. Querrec, "Semantic modeling of virtual environments using Mascaret," *In Software Engineering and Architectures for Realtime Interactive Systems (SEARIS), 2012 5th Workshop. IEEE.,* pp. 1-8, 2012, March.

[34] R. Atkinson, "Optimizing learning from examples using animated pedagogical agents," *Journal of Educational Psychology,* 2002.

[35] C. Hoareau, F. Ganier, R. Querrec, F. Corre and C. Buche, "Evolution of cognitive load when learning a procedure in a virtual environment for training," *6th International Cognitive Load Theory Conference (ICLTC'13),* 2013.

[36] R. Aylett and M. Cavazza, "Intelligent Virtual Environments-A state-of-the-art

report," *In Eurographics Conference, Manchester, UK.*, 2001, September.

[37] N. Gavish, T. G. Seco, S. Webel, J. Rodriguez, M. Peveri and U. Bockholt, "Transfer of skills evaluation for assembly and maintenance training," *In BIO Web of Conferences,* vol. 1, p. 28, 2011.

[38] J. A. Stevens and J. P. Kincaid, "The relationship between presence and performance in virtual simulation training," *Open Journal of Modelling and Simulation*, vol. 3(02), p. 41, 2015.

[39] C. Barot, D. Lourdeaux and D. Lenne, "Dynamic Scenario Adaptation Balancing Control, Coherence and Emergence," *ICAART*, pp. 232-237, 2013, February.

[40] K. Carpentier, D. Lourdeaux and I. M. Thouvenin, "Dynamic Selection of Learning Situations in Virtual Environment," *ICAART,* pp. 101-110, 2013, February.

[41] F. Le Corre, C. Fauvel, C. Hoareau, R. Querrec and C. Buche, "Chrysaor: an agent-based intelligent tutoring system in virtual environment," *International Conference on Virtual Learning*, 2012.

[42] J. Saunier, M. Barange, B. Blandin and R. Querrec, "A methodology for the design of pedagogically adaptable learning environments," *International Journal of Virtual Reality 16*, vol. 1, 2016.

[43] G. Rimassa, "Runtime support for distributed multi-agent systems," *Doctor degree thesis elaborated by Giovanni Rimassa at University Degli Studi de Parma,* 2003.

[44] V. Guéraud, J. M. Adam, J. P. Pernin, G. Calvary and J. P. David, "L'exploitation d'Objets Pédagogiques Interactifs à distance: le projet FORMID," *Revue des Sciences et Technologies de l'Information et de la Communication pour l'Education et la Formation (STICEF),* 2004.

[45] A. Newell, "Unified theories of cognition," *Harvard University Press*, 1994.

[46] P. Langley and D. Choi, "A unified cognitive architecture for physical agents," in *In Proceedings of the National Conference on Artificial Intelligence (Vol. 21, No. 2, p. 1469)*, Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press;, 2006.

[47] A. Newell, *Unified theories of cognition,* Cambridge, MA: Harvard University Press, 1990.

[48] J. R. Anderson, D. Bothell, M. D. Byrne, S. Douglass, C. Lebiere and Y. Qin, "An integrated theory of the mind," *Psychological review*, vol. 111(4), no. 1036, 2004.

[49] J. Lefevre, "J. Lefevre. Processing instructional texts and examples," *Canadian journal of psychology*, vol. 41(3), p. 351–364, 1987.

[50] P. Langley, J. E. Laird and S. Rogers, "Cognitive architectures: Research issues and challenges," *Cognitive Systems Research*, pp. 141-160, 2009.

[51] M. E. Bratman, D. J. Israel and M. E. Pollack, "Plans and resource-bounded practical reasoning," *Computational intelligence*, vol. 4(3), pp. 349-355, 1988.

[52] A. S. Rao and M. P. Georgeff, "Modeling rational agents within a BDI-architecture," *KR 91,* pp. 473-484, 1991.

[53] A. Guerra-Hernández, A. E. Fallah-Seghrouchni and H. Soldano, "Learning in BDI multi-agent systems," *Computational logic in multi-agent systems*. *Springer Berlin Heidelberg*, pp. 218-233, 2004.

[54] H.-Q. Chong, A.-H. Tan and G.-W. Ng, "Integrated cognitive architectures: a survey," *Artificial Intelligence Review 28.2*, pp. 103-130, 2007.

[55] M. Courgeon and C. Clavel, "MARC: a framework that features emotion models for facial animation during human–computer interaction," *Journal on Multimodal User Interfaces 7.4*, pp. 311-319, 2013.

[56] M. P. Georgeff and F. F. Ingrand, "Decision-making in an embedded reasoning system," Australian Artificial Intelligence Institute, 1989, pp. 972-978.

[57] M. d'Inverno, M. Luck, M. Georgeff, D. Kinny and M. Wooldridge, "The dMARS architecture: A specification of the distributed multi-agent reasoning system," *Autonomous Agents and Multi-Agent Systems*, vol. 9, pp. 5-53, 2004.

[58] M. Wooldridge, An introduction to multiagent systems, John Wiley & Sons, 2009.

[59] E. Norling, "Folk psychology for human modelling: Extending the BDI

paradigm," *In Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems - IEEE Computer Society,* vol. 1, pp. 202-209, 2004.

[60] A. Pokahr, L. Braubach and W. Lamersdorf, "Jadex: A BDI reasoning engine," *Multi-agent programming*. *Springer US,* pp. 149-174, 2005.

[61] J. E. Laird, A. Newell and P. S. Rosenbloom, "Soar: An architecture for general intelligence," *Artificial intelligence 33.1*, pp. 1-64, 1987.

[62] J. F. Lehman, J. Laird and P. Rosenbloom, "A Gentle Introduction to SOAR, an Architecture for Human Cognition," University of Michigan, 2006.

[63] A. M. Nuxoll and J. E. Laird, "Extending cognitive architecture with episodic memory," *Ann Arbor 1001*, pp. 1560-1564, 2007.

[64] M. J. Wooldridge, "Reasoning about rational agents," *MIT press,* 2000.

[65] J. R. Anderson and C. Lebiere, "The Newell test for a theory of cognition," *Behavioral and brain Sciences,* vol. 26(05), pp. 587-601, 2003.

[66] J. R. A. L. Anderson and C. Lebiere, "The atomic components of thought," *Lawrence Erlbaum*. *Mathway, NJ*., 1998.

[67] J. P. Borst and J. R. Anderson, "Using the ACT-R Cognitive Architecture in combination with fMRI data," *In An introduction to model-based cognitive neuroscience, Springer New York,* pp. 339-352, 2015.

[68] M. E. Foster, "Enhancing human-computer interaction with embodied conversational agents," *Universal access in human-computer interaction*. *Ambient interaction*. *Springer Berlin Heidelberg,* pp. 828-837, 2007.

[69] W. R. Swartout, J. Gratch, R. W. Hill Jr, E. M. Hovy, R. J. S. and D. Traum, "Toward virtual humans," *AI Magazine,* vol. 27(2), no. 96, 2006.

[70] J. F. Morie, E. Chance, K. Haynes and D. Rajpurohit, "Embodied conversational agent avatars in virtual worlds: Making today's immersive environments more responsive to participants," *Believable bots*. *Springer Berlin Heidelberg*, pp. 99-118, 2013.

[71] S. Kopp, L. Gesellensetter, N. C. Krämer and I. Wachsmuth, "A conversational

agent as museum guide–design and evaluation of a real-world application," *In Intelligent virtual agents, Springer Berlin Heidelberg,* pp. 329-343, 2005.

[72] A. Hartholt, D. Traum, S. C. Marsella, A. Shapiro, G. Stratou, A. Leuski, L.-P. Morency and J. Gratch, "All together now. Introducing the Virtual Human Toolkit," *Springer Berlin Heidelberg,* pp. 368-381, 2013.

[73] I. Poggi, C. Pelachaud, F. d. Rosis, V. Carofiglio and B. D. Carolis, "Greta. a believable embodied conversational agent," *Multimodal intelligent information presentation, Springer Netherlands,* pp. 3-25, 2005.

[74] J. Rickel and W. L. Johnson, "Animated agents for procedural training in virtual reality: Perception, cognition, and motor control," *Applied artificial intelligence,* Vols. 13(4-5), pp. 343-382, 1999.

[75] B. Hartmann, M. Mancini and C. Pelachaud, "Formational parameters and adaptive prototype instantiation for MPEG-4 compliant gesture synthesis," *In Computer Animation, 2002. IEEE,* pp. 111-119, 2002.

[76] C. Pelachaud, "Greta: an interactive expressive embodied conversational agent," *In Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems,* p. 5, 2015, May.

[77] M. Mancini, R. Niewiadomski, E. Bevacqua and C. Pelachaud, "Greta: a SAIBA compliant ECA system," *In Troisiéme Workshop sur les Agents Conversationnels Animés,* 2008.

[78] K. R. Thórisson, T. List, C. Pennock and J. DiPirro, "Whiteboards: Scheduling blackboards for semantic routing of messages & streams," *In AAAI-05 Workshop on Modular Construction of Human-Like Intelligence,* pp. 8-15, 2005.

[79] R. Niewiadomski, E. Bevacqua, M. Mancini and C. Pelachaud, "Greta: an interactive expressive ECA system," *In Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems. International Foundation for Autonomous Agents and Multiagent Systems,* vol. 2, pp. 1399-1400, 2009.

[80] P. Kenny, A. Hartholt, J. Gratch, W. Swartout, D. Traum, S. Marsella and D. Piepol, "Building interactive virtual humans for training environments," *In*

Proceedings of i/itsec, vol. 174, 2007, November.

[81] G. Littlewort, J. Whitehill, T. Wu, I. Fasel, M. Frank, J. Movellan and M. Bartlett, "The computer expression recognition toolbox (CERT)," *In Automatic Face & Gesture Recognition and Workshops (FG 2011), 2011 IEEE International Conference*, pp. 298-305, 2011, March.

[82] T. Bickmore, D. Schulman and G. Shaw, "Dtask and Litebody: Open source, standards-based tools for building web-deployed embodied conversational agents," *In International Workshop on Intelligent Virtual Agents, Springer Berlin Heidelberg*., pp. 425-431, 2009, September.

[83] M. Stone, "Knowledge representation for language engineering," *In A Handbook for Language Engineers*, 2003.

[84] P. Taylor, A. W. Black and R. Caley, "The architecture of the Festival speech synthesis system," 1998.

[85] A. Leuski and D. R. Traum, "NPCEditor: A Tool for Building Question-Answering Characters," *In LREC*, 2010, May.

[86] "VHToolkit - Confluence Institute for Creative Technologies," Confluence.ict.usc.edu, [Online]. Available: https://confluence.ict.usc.edu/display/VHTK.

[87] M. Courgeon, J. C. Martin and C. Jacquemin, "Marc: a multimodal affective and reactive character," *In Proceedings of the 1st Workshop on AFFective Interaction in Natural Environments*, p. 20, 2008.

[88] M. Courgeon, "MARC Toolkit," 2015. [Online]. Available: http://www.marc-toolkit.net/.

[89] A. Cafaro, H. H. Vilhjálmsson, T. Bickmore, D. Heylen and C. Pelachaud, "Representing communicative functions in saiba with a unified function markup language," *Intelligent Virtual Agents*. *Springer International Publishing*, 2014.

[90] S. Kopp, B. Krenn, S. Marsella, A. N. Marshall, C. Pelachaud, H. Pirker, K. R. Thórisson and H. Vilhjálmsson, "Towards a common framework for multimodal generation: The behavior markup language," *In International Workshop on Intelligent Virtual Agents, Springer Berlin Heidelberg*, pp. 205-217, 2006.

[91] H. Vilhjálmsson, N. Cantelmo, J. Cassell, N. Chafai, M. Kipp, S. Kopp, M. Mancini, S. Marsella, A. Marshall, C. Pelachaud and Z. Ruttkay, "The behavior markup language: Recent developments and challenges," *Intelligent virtual agents*. *Springer Berlin Heidelberg*, 2007.

[92] D. Heylen, S. Kopp, S. C. Marsella, C. Pelachaud and H. Vilhjálmsson, "The next step towards a function markup language," *Intelligent Virtual Agents*. *Springer Berlin Heidelberg*, 2008.

[93] S. Kopp, B. Jung, N. Lessmann and I. Wachsmuth, "Max-a multimodal assistant in virtual reality construction," *KI*, vol. 17(4), p. 11, 2003.

[94] M. Mancini and C. Pelachaud, "The fml-apml language," *In Proc*. *of the Workshop on FML at AAMAS*, vol. 8, 2008, April.

[95] R. Niewiadomski, M. Obaid, E. Bevacqua, J. Looser, L. Q. Anh and C. Pelachaud, "Cross-media agent platform," *In Proceedings of the 16th international conference on 3D web technology, ACM*, pp. 11-19, 2011, June.

[96] T. Salamon, "Design of agent-based models," *Eva & Tomas Bruckner Publishing*, 2011.

[97] J. Taoum, B. Nakhal, E. Bevacqua and R. Querrec, "A Design Proposition for Interactive Virtual Tutors in an Informed Environment," *16th International Conference on Intelligent Virtual Agents (IVA 2016)*, vol. 10011, pp. 341-350, 2016, September.

[98] Intel, "Intel® RealSense™ SDK," Intel, [Online]. Available: https://software.intel.com/en-us/intel-realsense-sdk.

[99] M. d. G. B. Marietto, R. V. d. Aguiar, G. d. O. Barbosa, W. T. Botelho, E. Pimentel, R. d. S. França and V. L. d. Silva, "Artificial Intelligence Markup Language: A Brief Tutorial," *arXiv preprint arXiv:1307.3091*, 2013.

[100] F. Le Corre, C. Hoareau, F. Ganier, C. Buche and R. Querrec, "A Pedagogical Scenario Language for Virtual Learning Environment based on UML Meta-model".

[101] S. Rubio, E. Díaz, J. Martín and J. M. Puente, "Evaluation of subjective mental

workload: A comparison of SWAT, NASA-TLX, and workload profile methods," *Applied Psychology,* vol. 53(1), pp. 61-86, 2004.

[102] V. Gawron, Human Performance Measures Handbook, Erlbaum, Lawrence, Associates, Publishers, Mahwah, N.J., 2000.

[103] J. A. Taylor and R. B. Ivry, "The role of strategies in motor learning," *Annals of the New York Academy of Science,* vol. 1251(1), pp. 1-12, 2012.

[104] E. Wenger, Artificial intelligence and tutoring systems: computational and cognitive approaches to the communication of knowledge, Morgan Kaufmann, 1987.

[105] R. Nkambou, R. Mizoguchi and J. & Bourdeau, Advances in intelligent tutoring systems, Springer Science & Business Media., 2010.

[106] G. Nkambou, R. Gauthier and C. Frasson, "Un modèle de représentation des connaissances relatives au contenu dans un système tutoriel intelligent," *Sciences et Techniques Educatives,* vol. 4, pp. 299-330, 1997.

[107] D. Lourdeaux, F. P. and B. J.-M., "An intelligent tutorial agent for training virtual environments," in *5th world multiconference on Systemics, Cybernetics and Informatics*, Orlando, USA, 2001.

[108] G. Vigano, S. Mottura, D. Calabi and M. Sacco, "The virtual reality design tool: Case studies and interfacing open topics," *In virtual concept,* p. 364–371, 2003.

[109] S. Pesty and C. Webber, "The baghera multiagent learning environment," *An educational community of artificial and human agents*, 2004.

[110] C. dos Santos and F. Osorio, "Integrating intelligent agents, user models, and automatic content categorization in a virtual environment," *ITS 2004, LNCS 3220,* p. 128–139, 2004.

[111] B. Sorensen and S. Ramachandran, "Simulation-based automated intelligent tutoring," *Human Interface, Part II, HCII 2007, LNCS 4558,* p. 466–474, 2007.

[112] C. Buche, C. Bossard, R. Querrec and P. Chevaillier, "PEGASE: A generic and adaptable intelligent system for virtual reality learning environments," *International Journal of Virtual Reality 9.2*, pp. 73-85, 2010.

8 APPENDICES

APPENDIX 1. FIPA PERFORMATIVES AND THEIR MEANING

Performative	When used
Accept Proposal	The action of accepting a previously submitted proposal to perform an action.
Agree	The action of agreeing to perform some action, possibly in the future.
Cancel	The action of one agent informing another agent that the first agent no longer has the intention that the second agent perform some action.
Call for Proposal	The action of calling for proposals to perform a given action.
Confirm	The sender informs the receiver that a given proposition is true, where the receiver is known to be uncertain about the proposition.
Disconfirm	The sender informs the receiver that a given proposition is false, where the receiver is known to believe, or believe it likely that, the proposition is true.
Failure	The action of telling another agent that an action was attempted but the attempt failed.
Inform	The sender informs the receiver that a given proposition is true.
Inform If	A macro action for the agent of the action to inform the recipient whether or not a proposition is true.
Inform Ref	A macro action for sender to inform the receiver the object which corresponds to a descriptor, for example, a name.
Not Understood	The sender of the act (for example, i) informs the receiver (for example, j) that it perceived that j performed some action, but that i did not understand what j just did. A particular common case is that i tells j that i did not understand the message that j has just sent to i.

Propagate	The sender intends that the receiver treat the embedded message as sent directly to the receiver, and wants the receiver to identify the agents denoted by the given descriptor and send the received propagate message to them.
Propose	The action of submitting a proposal to perform a certain action, given certain preconditions.
Proxy	The sender wants the receiver to select target agents denoted by a given description and to send an embedded message to them.
Query If	The action of asking another agent whether or not a given proposition is true.
Query Ref	The action of asking another agent for the object referred to by a referential expression.
Refuse	The action of refusing to perform a given action, and explaining the reason for the refusal.
Reject Proposal	The action of rejecting a proposal to perform some action during a negotiation.
Request	The sender requests the receiver to perform some action. One important class of uses of the request act is to request the receiver to perform another communicative act.
Request When	The sender wants the receiver to perform some action when some given proposition becomes true.
Request Whenever	The sender wants the receiver to perform some action as soon as some proposition becomes true and thereafter each time the proposition becomes true again.
Subscribe	The act of requesting a persistent intention to notify the sender of the value of a reference, and to notify again whenever the object identified by the reference changes.

APPENDIX 2. PARSING RULES OF FIPA-SL IN ANTLR

```
grammar FipaSL;

options {
    language=CSharp2;
}
@header {
   using System;
   using System.Collections;
   using System.Collections.Generic;
}
@members {
    public bool isAction = false;
    public bool isIota = false;
    public bool isEqual = false;
    public bool done = false;
    public bool started = false;
    public string value = "";
    public string performer = "";
    public string entityName = "";
    public string actionName = "";
    public string askedTerm = "";
    public string predicateSymbol = "";
    public List<string> paramValue = new List<string>();
    public List<string> paramName = new List<string>();
}
```

```
/*------------------------------------------------------------------
 * PARSER RULES
 *------------------------------------------------------------------*/
 content : LPAREN contentexpression RPAREN ;
 contentexpression : identifyingexpression
            | actionexpression
            | proposition ;
 proposition : wff;
 wff : LPAREN actionop actionexpression RPAREN
           | atomicformula;
 atomicformula : propositionsymbol
          | LPAREN predicatesymbol {predicateSymbol = $predicatesymbol.text;}
term+ RPAREN
          | LPAREN binarytempop term term {value=$term.text;} RPAREN
          | 'true'
          | 'false';
 actionop : 'done' {done = true;}
            | 'feasible'
            | 'started' {started = true;};
 term : constant
            | identifyingexpression
            | variable;
 binarytempop : '=' {isEqual = true;}
            | '>'
            | '>='
```

```
        | '<'
        | '<='
        | '!='
        | 'member'
        | 'contains'
        | 'result';
identifyingexpression : LPAREN referentialoperator {isIota = true;} term {askedTerm = $term.text;} wff RPAREN;
referentialoperator : 'iota';
actionexpression : LPAREN 'action'{isAction = true;} agent {performer = $agent.text;} functionalterm RPAREN;
functionalterm : LPAREN functionsymbol {actionName = $functionsymbol.text;} (parameter)* RPAREN;
parameter : parametername parametervalue ;
parametername : COLON ID  {paramName.Add($ID.text);};
parametervalue : term  {paramValue.Add($term.text);};
constant : numericalconstant
                | ID;
variable : variableidentifier;
variableidentifier : QUES ID {paramName.Add($ID.text);};
numericalconstant : integer
                | float;
agent : ID;
predicatesymbol : ID;
propositionsymbol : ID;
functionsymbol : ID;
```

```
integer : DIGIT+;

float : DIGIT+'.'DIGIT+;

DIGIT  : [0-9] ;

ID : [a-zA-Z_]([a-zA-Z_-]|[0-9])*;

WS : [ \t\r\n]+ -> skip;

LPAREN : '(' ;

RPAREN : ')' ;

COLON  : ':' ;

QUES : '?' ;
```

APPENDIX 3. THE PROCEDURE OF BLOOD ANALYSIS TESTS ON AN AUTOMATON BIOLOGICAL ANALYSIS MACHINE

	Action to realize	**Object to manipulate**	
1	Open Neoplastine	Tube Neoplastine	
2	Open Solvant	Tube Solvant	
3	Take Neoplastine	Tube Neoplastine	
4	Take Solvant	Tube Solvant	
5	Pour Solvant Neoplastine	Tube Solvant	Tube Neoplastine
6	Put Solvant	Tube Solvant	
7	Screw Neoplastine	Tube Neoplastine	
8	Shake Neoplastine	Tube Neoplastine	
9	Poser Neoplastine	Tube Neoplastine	
10	Open Neoplastine	Tube Neoplastine	
11	Open DesorbU	Tube DesorbU	
12	Take Maxi Reducer	Maxi Reducer	
13	Insert Maxi Reducer in DesorbU	Maxi Reducer	Tube DesorbU
14	Press Drawer Button (open)	Drawer Button	
15	Take Neoplastine	Tube Neoplastine	
16	Scanner Neoplastine	Tube Neoplastine	Scanner
17	Put Neoplastine in Drawer	Tube Neoplastine	Product Drawer
18	Take DesorbU	Tube DesorbU	
19	Scan DesorbU	Tube DesorbU	Scanner
20	Put DesorbU in Drawer	Tube DesorbU	Product Drawer
21	Press Drawer Button (close)	Drawer Button	
22	Take Tube	Tube	

23	Put Tube in Rack	Tube	Rack
24	Take Rack	Rack	
25	Put Rack in Basket	Rack	Basket
26	Take Basket	Basket	
27	Put Basket on STAR	Basket	STAR
28	Press Start Button	Start Button	
29	Take Basket	Basket	
30	Put Basket on Paillasse	Basket	Paillasse
31	Take Rack	Rack	
32	Put Rack on Paillasse	Rack	Paillasse
33	Press Drawer Button (open)	Drawer Button	
34	Take Neoplastine	Tube Neoplastine	
35	Put Neoplastine on Paillasse	Tube Neoplastine	Paillasse
36	Take DesorbU	Tube DesorbU	
37	Put DesorbU on Paillasse	Tube DesorbU	Paillasse
38	Press Drawer Button (close)	Drawer Button	

Génération des Intentions Communicatives pour Agents Virtuels dans un Environnement Virtuel Intelligent : Application aux Environnements d'Apprentissage Virtuels

Résumé

La réalité virtuelle joue un rôle majeur dans le développement de nouvelles technologies de l'éducation, et permet de développer des environnements virtuels pour l'apprentissage, dans lesquels, des agents virtuels intelligents jouent le rôle de tuteur. Ces agents sont censés aider les utilisateurs humains à apprendre et appliquer des procédures ayant des objectifs d'apprentissage prédéfini dans différents domaines. Nous travaillons sur la construction d'un système temps-réel capable d'entamer une interaction naturelle avec un utilisateur dans un Environnement d'Apprentissage Virtuel (EAV). Afin d'implémenter ce modèle, nous proposons d'utiliser MASCARET (Multi-Agent System for Collaborative, Adaptive & Realistic Environments for Training) comme modèle d'Environnement Virtuel Intelligent (EVI) afin de représenter la base de connaissances des agents, et de modéliser la sémantique de l'environnement virtuel et des activités des utilisateurs. Afin de formaliser l'intention des agents, nous implémentons un module cognitif dans MASCARET inspiré par l'architecture BDI (Belief-Desire-Intention) qui nous permet de générer des intentions de haut-niveau pour les agents. Dans notre modèle, ces agents sont représentés par des Agents Conversationnels Animés (ACA), qui sont basés sur la plateforme SAIBA (Situation, Agent, Intention, Behavior, Animation). Les agents conversationnels de l'environnement ont des intentions communicatives qui sont transmises à l'utilisateur via des canaux de communication naturels, notamment les actes communicatifs et les comportements verbaux et non-verbaux. Afin d'évaluer notre modèle, nous l'implémentons dans un scénarios pédagogique concret pour l'apprentissage des procédures d'analyse de sang dans un laboratoire biomédical. Nous utilisons cette application afin de réaliser une expérimentation et une étude pour valider les propositions de notre modèle. L'hypothèse de notre étude est de supposer que la présence d'un ACA dans un Environnement Virtuel (EV) améliore la performance du processus d'apprentissage (ou qu'au moins, ça ne le dégrade pas) dans le contexte de l'apprentissage d'une procédure spécifique. La performance de l'utilisateur est représentée par le temps requis pour l'exécution de la procédure, le nombre d'erreurs commises et le nombre de demande d'assistance. Nous analysons les résultats de cette évaluation, ce qui confirme partiellement l'hypothèse de l'expérience et affirme que la présence de l'ACA dans l'EV ne dégrade pas la performance de l'apprenant dans le contexte de l'apprentissage d'une procédure.

Mots clés : Environnement d'Apprentissage Virtuel, d'Environnement Virtuel Intelligent, MASCARET, Agents Conversationnels Animés, SAIBA, BDI, Intentions Communicatives.

Generation of Communicative Intention for Virtual Agents in an Informed Virtual Environment : Application to Virtual Learning Environment

Abstract

Virtual Reality plays a major role in developing new educational methodologies, and allows to develop virtual environments for learning where intelligent virtual agents play the role of tutors. These agents are expected to help human users to learn and apply domain-specific procedures with predefined learning outcomes. We work on building a real-time system able to sustain natural interaction with the user in a Virtual Learning Environment (VLE). To implement this model, we propose to use the Multi-Agent System for Collaborative, Adaptive & Realistic Environments for Training (MASCARET) as an Intelligent Virtual Environment (IVE) model that provides the knowledge base to the agents and model the semantic of the virtual environment and user's activities. To formalize the intention of the agents, we implement a cognitive module within MASCARET inspired by BDI (Belief-Desire-Intention) architecture that permits us to generate high-level intentions for the agents. Furthermore, we integrate Embodied Conversational Agents (ECA), which are based on the SAIBA (Situation, Agent, Intention, Behavior, Animation) framework. The embodied agents of the environment have communicative intentions that are transmitted to the user through natural communication channels, namely the verbal and non-verbal communicative acts and behaviors of the ECAs. To evaluate our model, we implement it in a concrete pedagogical scenario for learning blood analysis procedures in a biomedical laboratory. We use this application to settle an experiment to validate the propositions of our model. The hypothesis of this experiment is to assume that the presence of an ECA in a Virtual Environment (VE) enhances the learning performance (or at least does not degrade it) in the context of a learning procedure. The performance is represented by the time of execution, the number of committed errors and the number of requests for assistance. We analyze the results of this evaluation, which partially confirms the hypothesis of the experiment and assure that having an ECA in the VLE does not degrade the performance of the learner in the context of a learning procedure.

Keywords : Virtual Learning Environment, Intelligent Virtual Environment, MASCARET, Embodied Conversational Agents, SAIBA, BDI, Communicative Intention.