COMMUNICATIVE AUTONOMOUS AGENTS

Angela Caicedo, Jean-Sébastien Monzani and Daniel Thalmann
Computer Graphics Lab, Swiss Federal Institute of Technolog y (EPFL), CH 1015 Lausanne, Switzerland

Key words: Agent control, communicative virtual humans, trust.

Abstract: We present a way to mix the lower control of agents with the high level specifications of their goals. This paper addresses various topics required to animate virtual humans in a distributed way such as combining primary actions into tasks, using verbal communication between virtual humans and directing them with high level orders. Our models have been tested into a multi-languages / multi-modules application as described below.

1. INTRODUCTION

During the last years, the entertainment industry have produced a lot of exciting movies, games or TV shows involving realistic virtual humans. However, most of the work is hardly designed by artists and these impressing animations still require huge efforts. Furthermore, since movies are now integrating more and more virtual humans, there is a need for authoring tools specifically decicated to *autonomous* agents animation. This has been clearly demonstrated by the famous Improv system [17] or similar commercial tools, such as Motion Factory's Motivate [15] or Virtools' NeMo [19]. Efforts are continuously spent in order to obtain more and more realism: the use of speech, better animation, improved autonomy contribute to go toward life-like characters. Target applications do not only include the entertainment industry, but any inhabited virtual world might benefit from this kind of work. For example, we are now working on a simulator into which policemen have to deal with panic situations, with virtual humans running all around: this kind of training into a virtual environment is a good test for realistic autonomous agents.

Unfortunately, the animation of a virtual human is not an easy process: it actually involves various topics such as: motion control, action selection and verbal communication. Consequently, the *integration* of these domains altogether is a motivating technical challenge. The work presented by Bindiganavale *et al.* [2] is a good illustration of this goal. Our research is focusing on the same topic, that is the animation of autonomous virtual humans which are able to communicate verbally as we do. We are now going to briefly summarise the contributions and previous research for these domains.

From the animator's point of view, it is difficult for one agent to handle concurrent motions at the same time: how can one walk while carrying a box and looking around? If we are able to do this everyday, the simulation of simultaneous gestures and motions is a particular research subject. Models have been proposed to deal with that, such as Granieri's Parallel Transition Networks [10]. For the specific case of gestures involved in virtual humans conversation, Cassel *et al* [8] studied an automatic generation of movements and facial expressions (during conversation), based on the content of the dialog itself.

Regarding realistic verbal communication, we also need some sound propagation models. While Funkhouser, Min and Carlbom [9] introduced interesting algorithms for fast rendering of sound occlusion and diffraction effects, we think that simpler models simulating sound within a room and taking almost no CPU time have many useful applications in social simulations. A good example would be the simulation of a party, with many people speaking at the same time, and background music disturbing them. Our model is able to simulate such situations, without high computational cost.

Finally, an autonomous agent has to select its actions by itself. Research has been driven by people from different areas: ethologists such as Tinbergen [20], and computer scientists such as Brooks [6], Maes [13] and Minsky [14] who lead the school of Behaviour-Based Artificial Intelligence (BBAI). Our model, as proposed in the BBAI, does not attempt to build models of the world, and the agent has to reevaluate its course of action on every slot of time. Some points are not directly addressed by the BBAI such as the interplay between internal factors (emotional levels) and external factors (common world situations). Other authors such as Travers [21] have modelled a behavioural system where the agents are described in terms of *if-then* rules. However, we show in this paper that a simple predicate approach is not sufficient for modelling complex human behaviours based on different levels of emotions.

We are now going to present briefly our system and the various components embedded into it. We will continue with in section 3 with the *agent's brain.* Finally we describe in section 4 the agent's brain implementation in LISP, before concluding.

2. AGENT COMMON ENVIRONMENT

We have developed a system called: the *Agent Common Environment* (ACE) which animates virtual humans able to perceive their shared environment, perform different motions and have facial expressions. It also provides an easy way to plug-ins different behavioural modules.

ACE understands a set of different commands to be able to control the simulations: (i) Creation and location of 3D objects, virtual humans, and smart objects [12], (ii) Performance of different motion motors and facial expression: playing key-frames animation, using inverse kinematics [1], walking actions, etc. (iii) Virtual human interactions with smart objects. And (iv) Query of perception pipelines for a given virtual human [4].

All these commands are easily accessible from Python scripts, where different behavioural libraries can be created and plugged into ACE. Those scripts are basically ensuring the low level 3D animation of the virtual humans, while the high level decisions and behaviours are selected by the external Intelligent Virtual Agent behavioural module (see section 5). Thanks to the available packages coming with Python, one can manage easily concurrent processes with threads (such as, walking while looking at something), while a TCP/IP connection is maintained between the scripts and the Intelligent Virtual Agent. We are now going to describe the Agent Common Environment in details.

2.1 Agent design philosophy

The behaviour of agents is decomposed into two modules: the low-level animation and the high-level decisions taking. As many 3D environments, ACE is mainly coded in C++ to ensure high performances. For convenient user-interaction, it also provides the **Python layer** which interprets on the fly commands and animates the virtual humans. Python is an all-purposes scripting langage that we have extended to fit our needs. More precisely, when the application is launched, a simple environment is created and displayed in a window, and a command shell is prompted, ready for entering commands in Python. ACE provides the basic commands for loading, moving, animating humans and objects, giving a powerful set of functionalities straight from the scripting language. It is very convenient indeed to reuse a language and extend it to match our purposes, rather than developing a new syntax from scratch: this saves time and gives the opportunity to reuse third-party modules, which have been already implemented and tested by others. On the other hand, the **Intelligent Virtual Agent** (IVA) is in charge of making decisions, e.g. choosing the next action to take place, deciding what are the new goals of the agent, managing the dynamics of the agent's emotions during the simulation, and so on. Information is stored here in an abstract way, leaving the high to low level binding to the Python layer. For instance, to indicate a specific furniture in an office, we will specify it as *the chair next to the window* rather than x, y and z coordinates: this mapping is handled directly in Python. To conclude, the IVA can be consider as the agent's *brain*.

2.2 Multiple inheritance architecture

Running into ACE, the script for each agent should handle various capabilities, such as: perception, verbal communication, performing actions and connecting to the IVA behavioural module. Thus, we split each capability into one class and merged all of them into the definition of what an agent should be able to do. Using UML [3], we present in *Figure 1* the definition of the **Agent** class, as implemented in Python.

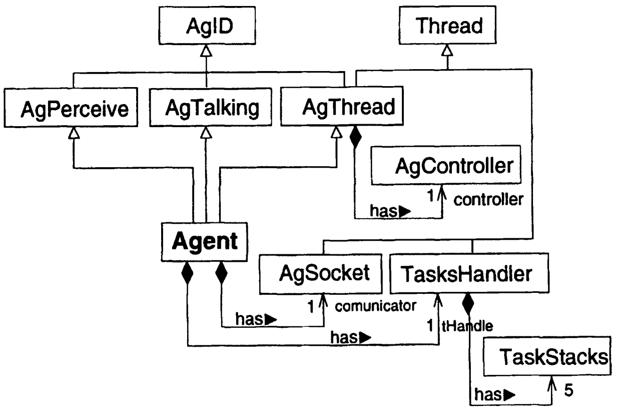

Figure 1. Multiple inheritance architecture defining one agent

Since each agent has a unique ID, we start by defining the AgID class as a super class, sharing the ID among the inherited classes. From this, we derive three basic classes, for the various capabilities, as pointed out before: the **AgPerceive** class encapsulates all the methods that allow the agent to visually perceive objects and remembers when objects get on/out of focus.

AgTalking lets the agent communicate by speaking to and hearing other agents. **AgThread** is the basic class for running one thread per agent, which means that each agent is running its own code in its thread (these functionalities are provided by the standard Thread class). Each thread is registered into an AgController which is then in charge of monitoring them. It also provides a shared space for exchanging information between the threads.

The final **Agent** class inherits from these three basic classes, which of course means that our **Agent** is able to speak to someone, hear when someone speaks and perceive the objects in the environment. But the **Agent** still needs to use some other modules: the **TasksHandler** which is in charge of handling parallel tasks like walking, looking, playing keyframes, applying facial expressions or interacting with objects and the **AgSocket**: each agent should be connected in some way to its IVA behavioural module and this is achieved by this class. The AgSocket class is able to decode orders coming for the IVA or send stimuli like visual perception back to the it. By using sockets and TCP/IP connection, the system can run in a distributed way, reducing the CPU cost on the machine which is responsible of the 3D environment display. The communication between the **Agent** object and the corresponding IVA is summarised in *Figure 2.*

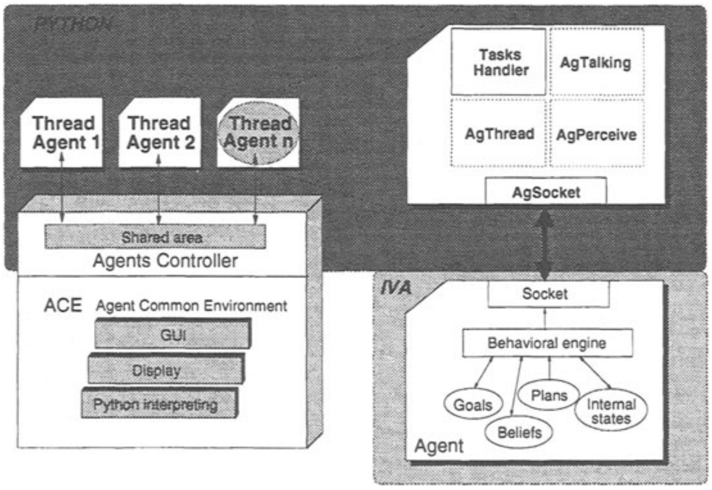

Figure 2. ACE system and connections to the Intelligent Virtual Agent (IVA)

2.3 The use of threads

One major improvement in adding the Python interpreter is the easy way of creating threads within it. Threads all run in parallel and efficient synchronisation primitives are available, such as events. This is a very convenient way to perform actions in parallel. Blocking actions such as waiting for data or event (for instance, a task to finish) could easily be handled by such threads. While it is very tempting to use threads to mimic human capabilities of performing various actions at the same time, one should take care of not creating too many threads (let's say, one per action), since it might take too much CPU time. That is why we are concerned in the next sections by simulating parallel behaviours within non-concurrent instructions too.

Our **Agent** has mainly three threads: the **Agent itself**, the **Tasks Handler**, and **the Agent Socket**. The main task of the **Agent** is to be alert of what he sees, or hears, and to give the appropriate response when one of these events happens. Even if the agent is managing socket connections and parallel tasks, it has not to worry about this matters, because this is continuously handled by separated threads. The **Tasks Handler** is a thread that is managing the stacked tasks performed or to be performed by the **Agent**. This thread is in charge of choosing the tasks that will be triggered in the next time slot. The **Agent Socket** monitors the activity of the socket, this means, is in charge of reading from the socket the incoming data, and writing the outgoing data or feedback data to the **IVA brain**.

3. INTERCONNECTING THE ANIMATION AND BEHAVIOURAL MODULES

As we have already mentioned earlier, the agent's animation in handled by Python scripts (and by the **Agent** class) while behaviour selection and decisions are chosen into the Intelligent Virtual Agent. Both are connected through sockets, and the **Agent Socket** (defined in Python)n is in charge of interconnecting the high level orders coming from the IVA with orders understandable by the **Agent** defined in Python, and vice-versa. We can basically distinguish three kinds of communications:

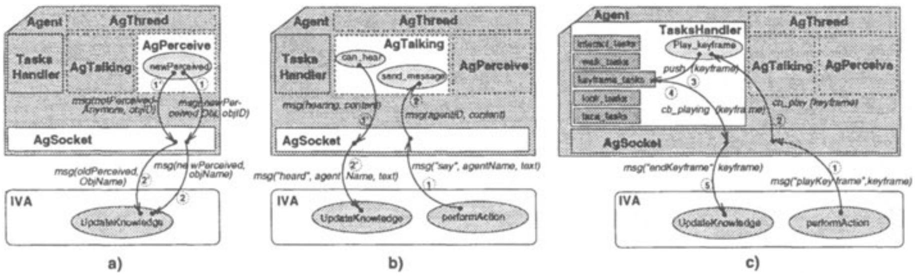

Figure 3. Communication between the Agent Python class and the IVA

1. **Perceiving an object or another agent**: whenever any new object is perceived, the method *newPerceived* inherited from AgPerceive returns *true*, and a message is created for the AgSocket (see *Figure 3a*). This message consists of a short description of what happened, and the ID of the perceived object. The AgSocket receives this message and translates it for the IVA brain, which finaly maps the ID to the corresponding object name. Similarly, the method *newPerceived* is also used to update the objects that are not visible anymore.
2. **Speaking to and hearing another agent**: when someone starts to speak, the method *can-hear* inherited from AgTalking returns *true*, and the **Agent** receives the incoming message. The *is-speaking* and the *end-of-message* messages are ignored, because these ones are just used for synchronisation purposes. The AgSocket again is in charge of extracting the relevant information for the IVA brain, and creates a new message that contains the name of the agent who spoke, and the utterance. The speaking process is a little bit different, because it is the IVA this time which starts the conversation, as presented in *Figure 3b*. The message consists of the action that will take place (in that case, the action *say*), the agent receiver's name, and the text that the agent wants to say. The AgSocket receives this message and generates three new *SpokenMessages*: *is-speaking*, *message-interchange* (which carries the semantic) and *end-of-message* to finish the communication [16].
3. **Walking, looking, playing keyframes or applying face actions**: these tasks are treated in the same way by the **Agent** in Python, specifically by the **Agent's Tasks Handler**. Again, the IVA brain triggers the need of performing one of these tasks, sending a message to the AgSocket, which then activates the corresponding **task callback** associated with the task and push it into its **Tasks Stack**. The **Tasks Handler** keeps checking for the termination callback of all the tasks inside the **Tasks Handler**, and when the termination callback is triggered, a new message is sent to **AgSocket** to reflect the changes into the **Agent's** brain (see *Figure 3c*).

4. THE IVA BRAIN: INTELLIGENT VIRTUAL AGENT

The Intelligent Virtual Agent is based on a BDI architecture (Beliefs, desires and intentions), widely described by Georgeff [18]. This architecture is promising but needs some extensions for achieving our goal: giving to the virtual human the ability to act by itself in a dynamic environment relying on its beliefs, internal states, current state of the surrounded world and assumptions about other agents. It should also allow us to control it in real time [7].

4.1 IVA's components

An IVA has all its knowledge organised into sets, which are distributed according to their functionality (*Figure 4*): the set of *Beliefs*, the set of *Goals*, the set of *Competing Plans*, the set of *Internal states*, the set of *Beliefs About Others*. Based on all its knowledge, the IVA is able to select the correct action to perform, in order to achieve its goal. This process is done by the *Behavioural Engine* which will be explained later in this paper.

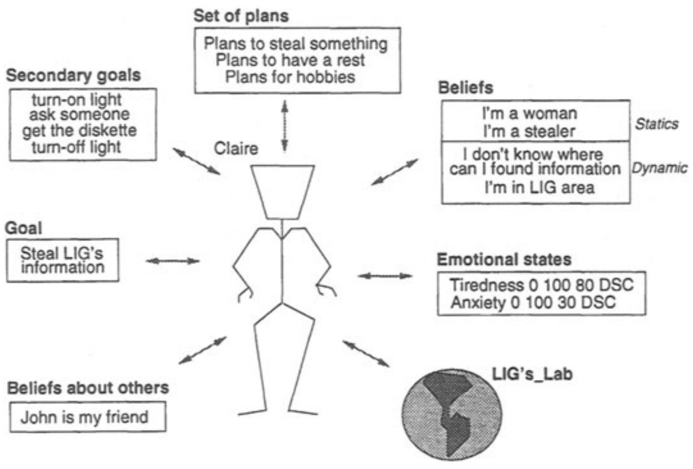

Figure 4. The Intelligent Virtual Agent (IVA)

1. **Beliefs** are a set of statements that the IVA believes to be true. The agent's beliefs are organised to let us simulate *short term memory* by the **Short term beliefs (STB)**, and *everlasting memory* by the **Long term beliefs (LTB)**.
2. IVAs have one main **Goal** and one or several **Subgoals**. The main goal is the objective that the IVA is trying to achieve at a certain moment. During this process, an IVA has to deal with smaller subgoals on which the outcome of the larger one relies on.
3. Internal states: The agent stores a set of internal states representing physiological or psychological variables of the virtual human. Internal state act as stimulus for the agent, i.e. *a high hunger level will stimulate the agent to eat*. An internal state is_i is described as a tuple: (n_i, min_i, max_i,c_i, cat_i), where for any given internal state *i*: n_i is its name, min_i is its minimum accepted value, max_i , the maximum accepted value, c_i

the current value, and cat_i is its category. Internal states are constantly being adjusted, as the simulation evolves and plans are adopted. Changes in the internal state are consequences of: the **autonomous growth or damping** associated with the internal state and the **side-effects** of an active behaviour. We categorise the internal states as ascendant (the higher the level the better), descendants and not categorised.

4. **Competing plans:** An IVA uses a set of competing plans that specified a sequence of actions required to reach its main goal. A competing plan P_i is described as: $P_i = (is_i, pc_i, ef_i)$, where: is_i is a list of internal states to be checked before the plan can be executed. Each of the internal states has an associated valid value or range. pc_i is a list of preconditions which have to be true before the competing plan can be triggered. The preconditions belong either to the agent's beliefs or to the general knowledge stored in the world. ef_i is a list which contains the effects of a plan execution. When a plan is selected, changes at agent or world level will occur (new knowledge will be added and old one will be deleted). These changes are consequences of the plan's effects.
5. **Beliefs about others:** In our model each IVA is autonomous, and can accept or reject an order coming from the user or from another agent. Each IVA includes a set of Beliefs about others into which it stores the trust levels associated with them. An IVA sees the user as another agent, and depending on the user's category it will accept an order or not. The levels of trust will evolve during the simulation [7], following the Hinde statement: *"Trust, once established in some degree, is often self-reinforcing because individuals have stronger tendencies to confirm their prior beliefs than to disprove them."* [11]. All IVAS contain the name of the other agents and the level of trust associated to them. The value of acceptance for any order coming from a user is handled so that the higher/lower the trust level, the higher/lower the possibility of accepting the order.

4.2 The Behavioural Engine (BE)

The behavioural engine is in charge of updating the internal states of the IVA and selecting its next action. It is composed of some controllers as shown in *Figure 5*. First the *Event Controller* checks in the pending events list for those events that trigger in a specific time slot to be integrated in the IVA's knowledge. Then the *Plan Seeker* sequentially passes the plans to the *Plan Controller* which verifies if the plan will be trigger or not. A plan to be triggered needs to have the suitable internal states levels and to full-fill all the preconditions. The *State Controller* checks the internal states levels and if all of them have the appropriate values it will give the control to the

Precondition Controller, otherwise the *Plan Seeker* will search for the next plan to evaluate. The *Precondition Controller* searches if all the preconditions are full-filled from its local knowledge, or from the external knowledge (*World Agent*). If the *Precondition Controller* agrees with all the preconditions the *Effects Performer* will be called, in order to perform all the necessaries updates inside the IVA or in the *World Agent*, and send the selected action (if there is one) to the *Virtual Human*.

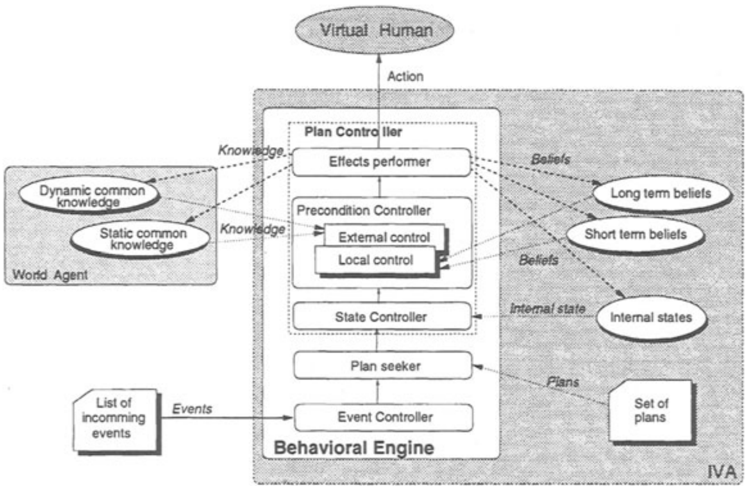

Figure 5. Behavioural Engine

5. CONCLUSION

We have presented in this paper various requirements to go toward life-like agents: our system has a multi-layered and distributed multi-languages architecture. We used **Tasks** to combine primary actions altogether, and we have presented a model for simulating verbal communication. The high level IVA brain, independent of graphics specification, is able to intelligently interact with a lower level module to create one single unit: the *Agent*.

REFERENCE

[1] P. Baerlocher and R. Boulic. Task priority formulations for the kinematic control of highly redundant articulated structures. In *IEEE IROS' 98*, pages 323–329, 1998.

[2] R. Bindiganavale, W. Schuler, Allbeck J., Badler N., Joshi A., and M. Palmer. Dynamically altering agent behaviors using natural language instructions. In *Autonomous Agents 2000 Proceedings*, 2000.

[3] Grady Booch, Ivar Jacobson, James Rumbaugh, and Jim Rumbaugh. *The Unified Modeling Language User Guide*. Addison-Wesley, 1998.

[4] C. Bordeux R. Boulic and D. Thalmann. An efficient and flexible perception pipeline for autonomous agents. In *Proceedings of Eurographics' 99*,.

[5] R. Boulic, P. Becheiraz, L. Emering, and D. Thalmann. Integration of motion control techniques for virtual human and avatar real-time animation. *ACM Symposium on Virtual Reality Software and Technology*, September 1997.

[6] R. Brooks. A robust layered control system for a modbile robot. *IEEE Journal of Robotics and Automation RA-2*, 1986.

[7] A. Caicedo and D. Thalmann. Virtual humanoids: let them be autonomous without losing control. In *The Fourth International Conference on Computer Graphics and Artificial Intelligence*, 2000.

[8] J. Cassell, C. Pelachaud, N. Badler, M. Steedman, B. Achorn, T. Bechet, B. Douville, S. Prevost, and M. Stone. Animated conversation: Rule-based generation of facial expression gesture and spoken intonation for multiple.*Proceedings of SIGGRAPH 94*

[9] Thomas A. Funkhouser, Patrick Min, and Ingrid Carlbom. Real-time acoustic modeling for distributed virtual environments. *Proceedings of SIGGRAPH 99*

[10] J. P. Granieri, W. Becket, B. D. Reich, J. Crabtree, and N. L. Badler. Behavioral control for real-time simulated human agents. *1995 Symposium on Interactive 3D Graphics*

[11] Robert Hinde and Jo Groebel. Cooperation and prosocial behaviour. Cambridge University Press, 1991.

[12] M. Kallmann and D. Thalmann. A behavioral interface to simulate agent-object interactions in real-time. In IEEE Computer Society Press, editor, *Proceedings of Computer Animation 99*, pages 138–146, 1999.

[13] P. Maes. How to do the right thing. *Connection Science Journal*, 1:291–323, Dec 1989.

[14] M. Minsky. *The society of mind*. Simon and Schuster, 1988.

[15] Karen Moltenbrey. All the right moves. *Computer Graphics Word*, 22, October 1999.

[16] J.-S. Monzani and D. Thalmann. Verbal communication: Using approximate sound propagation. In *Autonomous Agents'2000 Conference Proceedings*, 2000.

[17] Ken Perlin and Athomas Goldberg. Improv: A system for scripting interactive actors in virtual worlds. *Proceedings of SIGGRAPH 96*, pages 205–216, August 1996.

[18] A. S. Rao and M. P. Georgeff. Modeling rational agents withing a bdi-architecture. *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning*. Morgan Kaufmann, 1991.

[19] Dan Teven. Virtools' NeMo. *Game Developer Magazine*, September 1999.

[20] N. Tinbergen. *The study of Instinc*. Oxford University Press, 1951.

[21] M. Travers. *Agar: An animal construction kit*. PhD thesis, The Media Lab, MIT, 1988.