A Layered Approach to Animate Intelligent Characters

Ana Paula Cláudio Graça Gaspar Luís Moniz Maria Beatriz Carmo Ricardo Abreu

Daniel Policarpo Marco Lourenço Nuno Martins

Departamento de Informática
Faculdade de Ciências da Universidade de Lisboa
apc@di.fc.ul.pt; gg@di.fc.ul.pt; hal@di.fc.ul.pt; bc@di.fc.ul.pt; gumbeto@gmail.com
policarpodan@gmail.com; emacs.me@gmail.com; nuno.a.d.martins@gmail.com

Abstract
The IViHumans platform supports the development of diverse applications with virtual humans. It comprises one layer for graphical processing and one for artificial intelligence. The layers were projected to run in different processes, communicating by means of a simple, yet effective and extensible client/server protocol that we projected and implemented. In this framing, the graphical processing layer plays the role of server, while the artificial intelligence layer occupies the position of the client. Therefore, the graphical processing layer is the base of the platform, providing services for the intelligent agents that populate the artificial intelligence layer.

Keywords
Virtual humans, Virtual environments, Synthetic Perception, Steering behaviors, Multi-agent systems

1. INTRODUCTION

Virtual environments that are inhabited by agents with a human embodiment have many practical applications nowadays in areas such as entertainment, education, psychotherapy, industrial training and reconstitution of historical places. We are engaged in a research project that aims at building a flexible library for materializing a system of multiple intelligent agents into Virtual Humans (VHs) that inhabit arbitrary virtual environments – the IViHumans (Intelligent Virtual Humans) platform [Abreu08].

The IViHumans platform is being developed to accommodate VHs that combine a realistic appearance with a performance that approximates human cognition and behavior. To pursuit this goal, the platform is composed of two separate layers: one for Graphical Processing (GP) and one for Artificial Intelligence (AI) computation. These layers express the control of agents at different detail levels.

The following section summarizes the fundamental related work, stating the main similarities and differences of our approach. In section 3 we present the general architecture of the IViHumans platform. In section 4 a brief general description of the GP Layer is presented. In section 5 the communication between the two layers is presented. In section 6 we present the AI Layer, describing the several types of agents that constitute it. The last section summarizes this work and discusses some future work.

2. RELATED WORK

In the last decade some other authors have presented work integrating multi-agent systems with graphics components. In [Barella06] a game-oriented multi-agent system built over the JADE multi-Agent System (MAS) platform is presented. That approach is similar to ours, in that the MAS module and the visualization module work independently of each other. However in their system the simulation is centralized in the MAS, while the visualization module has the sole duty of exhibiting the world without allowing interaction.

Several other authors have used Belief-Desire-Intention (BDI) agent platforms together with game engines or specific visualization platforms. Some of those works concentrate on particular types of applications and agents. For instance, in [Magerko04] an environment based on the Unreal Tournament game engine and the Soar AI engine is described, concentrating in the development of complex AI agents for interactive drama applications. In [Norling04] BDI agents that model expert players of Quake 2 are developed.

In [Torres03, Torres04] the BDI model is also used to control animated characters in virtual worlds. In their work, 3D articulated characters are controlled in real time by cognitive agents that are clients of the environment which, in its turn, acts as a server and is responsible for managing the information that can be perceived by the agents. However their architecture is limited in that everything an agent can perceive must be listed a priori in a list of boolean statements.

In [Norling01] the JACK agent language is extended with a specific perceptual/motor system that allows the agents to interact with a graphical interface. Our approach is instead to make the GP layer responsible for the perception and basic motion of the agents, thus decoupling the perceptual/motor system from the cognitive part of the agents and making it more independent of the agents' implementation.

In [Evertsz07] a military simulation tool, ROE3, is presented where the agents are also implemented using JACK. In ROE3, an Integration Layer translates messages between the agent and the synthetic environment, being able to aggregate raw percepts into higher level percepts. In this respect, our platform incorporates a similar translation mechanism, as part of the AI Layer, in the form of interface agents.

In what concerns virtual perception and movement, our approach was strongly influenced by the work of Craig Reynolds. In 1987 he released a distributed behavioral model for the simulation of animated flocks, herds, and schools [Reynolds87], breaking up with traditional approaches of scripting the movement of virtual actors and objects and introducing the behavioral animation concept. In the model he built, "boids" decide their routes autonomously and at runtime. The decision of each individual is achieved according to a set of behavioral rules and using simple models of local perception of a dynamic environment. In order to have a consistent behavior, boids act according to a limited awareness of the surrounding environment, which is obtained through what is often called synthetic perception.

Vosinakis et al. [Vosinakis01], Noser et al. [Noser95], and Peters and Sullivan [Peters02], all apply a model of virtual vision that acts by filtering out the objects that should not be seen by the characters, granting them access to the information that characterizes the ones that are seen (though sometimes its detail depends on attention). In the IViHumans platform, we use a model of raycasting synthetic vision, like Vosinakis et al. did. We do so with an original algorithm that tries to maximize precision without any meaningful impairment of efficiency and that provides parameters that can be tweaked to find the correct balance. Like these authors, we also have a memory model coupled with perception. It is able to remember default and custom properties of objects with three different memory management techniques. Our memory is capable of recalling the different states an object goes through along time, rather than just one instantaneous state. This allows the extraction of conclusions about the evolution of the world.

In later work, Craig Reynolds generalized his behavioral model to imbue different characters with realistic and spontaneous navigation capabilities [Reynolds99]. The rules for the movement of the boids were expressed using Steering Behaviors. Reynolds proposed a conceptual division of the behaviors of a virtual character in three separate layers, ranked by degree of abstraction:

- The locomotion layer includes low-level tasks that carry the movement of the character;
- The steering layer establishes the way to go to reach an objective;
- The action selection layer defines objectives and sub-objectives.

According to this hierarchical layout, the aims of the action selection layer are achieved through the features that the steering layer provides, and the same relation holds between the steering layer and the locomotion layer. In Reynolds' vision, steering behaviors are a part of the steering layer.

Steering behaviors stand on the assumption that the underlying layer can be approximated by a simplified concept of vehicle that can equally be applied to planes, cars, horses, or the legs of a human being (with some abuse of nomenclature, as the author points out).

We consider Reynolds' conception of virtual movement as a grounding base for our approach and we adapt and implement it in our own way. We pay special attention to locomotion and our characters display consistent animations that are automatically chosen according to their own rules.

Like Vosinakis et al. [Vosinakis01], Rickel et al. [Rickel97] and Longhi et al. [Longhi04], we also account for emotions in the IViHumans platform. Emotion will integrate cognitive models on the AI layer, according to what is described in [Morgado07]. On the GP layer, it will be indirectly expressed in the behavior of the characters and, directly, through their facial expressions. Unlike these authors, however, we propose a solution for facial expressions that allows the exhibition of complex expressions and that supports smooth transitions between them.

In the overall perspective, our architecture is also different from others. Torres et al. implement agents as individual processes. The agents play the role of clients of an environment that acts like server. Vosinakis et al. have the AI code supplied by the user through callback functions. The IViHumans platform seeks to evenly distribute computation tasks between both layers. The GP layer includes some behavioral features that are sometimes classified as belonging to the field of artificial intelligence [Buckland05]. This allows reactive behavior to happen faster, without having to wait for the decisions of the AI layer.

The IViHumans platform is projected to be generally applicable, in order to enable different applications to use and build upon it. Our aim is to allow user applications to focus and extend either one of the layers, or both. Interaction with the user can be included in either side, as both layers have a determinant influence in the course of events, unlike what happens in the works of Perlin et al. [Perlin96], Ulicny et al. [Ulicny01] and Barella et al. [Barella06]. Also, in the approaches of these authors, the server is on the artificial intelligence side, while in ours the server is on the GP layer. Agents operate according to the basic services that the graphical layer, which is at a lower level, provides, instead of having the latter with the sole responsibility of rendering the effects of agents' operations. In this sense, our view is similar to the one manifested by Funge et al. [Funge99], in the sense that cognition is built on top of lower-level functionality.

The IViHumans platform separates the GP layer from the AI layer, an approach that is already found in works such as the JGOMAS system [Barella06] and the ROE3 architecture [Buckland05]. However, our proposal differs from them by having the amount of responsibility of each layer well balanced. For instance, in what regards sensory honesty, the physical limits to what can be perceived by a character are controlled by the GP layer while the cognitive restrictions reside on the AI layer. Also, the GP layer is responsible for quickly handling aspects such as typified steering behaviors, while the AI layer deals with the more complex cognitive behavior, using symbolic representations. As a side effect, this architecture also tends to reduce considerably the communication overhead.

3. ARCHITECTURE

The IViHumans platform aims at a uniform balance of responsibility between the two layers. The GP layer is responsible for quickly handling the low-level aspects that mainly characterize graphical and physical processing, providing an abstracted interface to the AI layer that deals with the more complex cognitive behavior, using symbolic representations.

The GP layer is built on top of the rendering engine OGRE[1]. The Multi-Agent system – the core of the AI layer – is built upon the JADE[2] framework.

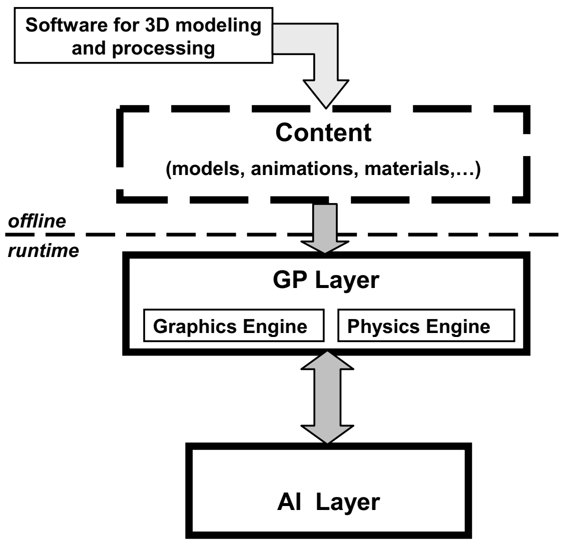

Software for 3D modeling and processing
Content
(models, animations, materials,…)
offline
runtime
GP Layer
Graphics Engine
Physics Engine
AI Layer

Figure 1: Content creation and architecture of the platform

[1] http://www.ogre3d.org/

[2] http://jade.tilab.com/

The main task for which the GP layer is responsible is the representation of all the elements contained in the virtual world, among which the VHs are the most important, reproducing the appropriate animations that carry the flow of occurrences and consistently enacting the evolution of the world and its components. The AI layer, on the other hand, hosts and manages the minds of the VHs, so that each one is controlled by one or more agents that entitle him with intelligent behavior. In the AI layer there are essentially two types of agents: interface agents and cognitive agents. The interface agents deal with the communication, transforming raw data into meaningful symbolic representations and translating high-level actions into the appropriate commands. Interfacing components are also present in the GP layer.

The communication between the two layers follows the client/server approach. The GP layer provides the server, and the AI layer connects to it through a client. Our server relies heavily on the boost libraries, mainly on Boost.Asio, an opensource cross-platform C++ library for network programming that provides developers with a consistent asynchronous I/O model using a modern C++ approach[3] .

This library is based on the Proactor design pattern, whose description can be found elsewhere [Schmidt00].

The layers of the IViHumans platform communicate with a very simple protocol that we built over TCP. The protocol is extensible because new messages can be added by any application that uses the platform, provided that the corresponding parsers and interpreters are included.

4. PERCEPTION, MOVEMENT, AND EMOTION EXPRESSION

The VHs are the central elements the IViHumans platform has to deal with and we put them at the heart of the GP layer's implementation. Concretely, the VHs are implemented, in the GP layer, by the class IViHuman. Objects of this class are particular VHs that can perceive their environment with synthetic vision, execute steering behaviors, and exhibit expressions.

4.1 Perception: synthetic vision and memories

Some approaches to games and simulations implement virtual agents with unrestricted access to the world data, bestowing them with sensory omnipotence [Buckland05].

These approaches incur in a flaw that originates unnatural events in the behavior of the intelligent agents. Their actions demonstrate they have knowledge they should not have, like being aware of objects in different rooms. This is not a good approach when believability is sought. It is essential to hold back from the agents the information they should not be capable of accessing, otherwise it will be apparent that the AI is “cheating”.

A VH perceives its environment through a ray-casting synthetic vision algorithm [Semião06b, Semião06a]. The algorithm allows the parameterization of several characteristics, which gives it flexibility. For instance, the right

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

equilibrium between accuracy and efficiency for a particular environment or purpose can be achieved by tweaking frequency (the rate at which the algorithm iterates), intensity (the number of rays that are cast at each iteration), or the displacement of rays over successive time steps. Each VH has a camera that inherits the movement of his head. This camera is used both as a means of displaying what the VH sees, and as a reference for deriving the direction of the rays that are cast into the scene. Other sensing mechanisms may also be added, without changing existing code, by implementing a simple interface.

The class SyntheticRayVision implements the vision algorithm. Synthetic vision can be explicitly called upon, to obtain information about the objects that a VH sees at any given moment. The AI layer can post such requests on the GP layer any time, and the vision algorithm will be executed. Memories and beliefs can then be managed by the intelligent agents. However, we included an alternative in the GP layer that releases the AI layer from the burden of constantly asking the VHs what they see. It may also be used for reactive behavior directly handled by the GP layer (e.g. by creating steering behaviors that use it). This alternative consists on an automatic vision mode with custom memory associated.

An IViHuman can be told to activate the “auto vision" mode, so that he periodically executes his vision algorithm on his own.

There would be no point in having the vision algorithm automatically executed if the results are not used in a short time period. As it is, the IViHuman class does not analyze the results of the vision algorithm, as this is assumed to be mainly a responsibility of the AI layer.

So, the auto vision mode cannot be executed unless the IViHuman has a memory that can record relevant data and provide it whenever needed. VisionMemory is the abstract class that represents this memory, as shown in Figure 2.

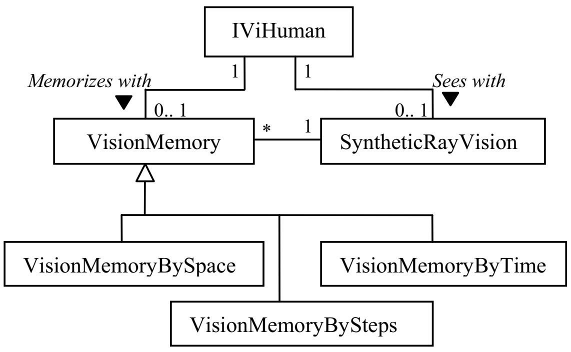


Figure 2: Class Diagram – Vision and Memory

Every VisionMemory instance is a listener of a SyntheticRayVision object and is notified whenever the vision algorithm is executed. A VisionMemory can be told what properties to get from the objects that are seen by its owner.

VisionMemory is only an abstract class that aggregates the main features that must be provided by every actual memory. We implemented three subclasses that store and manage data in their own way: VisionMemoryBySpace, VisionMemoryByTime, and VisionMemoryBySteps.

Even though they manage their storage space in distinct ways, all the implementations have some limit for their capacity, which is provided upon construction. Thus, they function only as short or medium term memories. A more persistent memory model should be built in the AI layer, possibly storing data after it is processed to a representation more adequate for reasoning and cognition.

VisionMemoryBySpace stores information for a certain number of objects. Its capacity is determined in terms of the number of object entries that can be stored at any given time. VisionMemoryByTime stores all the information obtained in a certain time span, which is specified by its capacity. An absolute timer is kept and updated when the object is notified of new vision results. VisionMemoryBySteps is very similar to VisionMemoryByTime, but it uses relative time instead of absolute time, as VisionMemoryBySpace does. It stores all the information obtained in a certain number of vision iterations.

The contents of the IViHuman's memory can be requested by the AI layer. With this memory model, agents can ask for the most recent value of some particular property, for a particular object or for all objects that were seen. They can trace how a certain property varied recently for some object or draw conclusions from relating properties of different objects. They can also request information, from time to time, to monitor the world while reasoning, or to build a persistent memory, at the pace that suits them best.

4.2 Movement

The motion of the VHs is supported by the notion of steering behavior, as explained before. We adopt Craig Reynold’s hierarchical categorization of movement in the three layers of locomotion, steering and action selection. The former two are under the domain of the GP layer, while the last and topmost layer can either be included in the AI layer, which happens for virtual humans, or absent, so that human users can directly control avatars. Locomotion corresponds to the choice of the appropriate animations on the basis of speed and according to the rules that are unique for each VH model. Some steering behaviors are already included but, again, many more can be implemented and immediately used. In what concerns movement, the action selection layer operates by activating and parameterizing steering behaviors to achieve the desired goals, according to agents’ planning.

The most obvious attributes that a VH must have are the ones that describe its basic physical state, such as position, velocity, or orientation. Values for these attributes can be derived by applying laws of classical physics. Using these laws, an IViHuman can update its own physical state if it is given the time that elapsed since the last state. However, this ability is not exclusive of a VH. It is part of every entity with physical existence that moves in the virtual world, be it a ball, a car, or a camera. Thus, we created a base class that aggregates this functionality, so that every moving entity can inherit from it, and we called it IViEntity. Its inheritance diagram is shown in Figure 3.

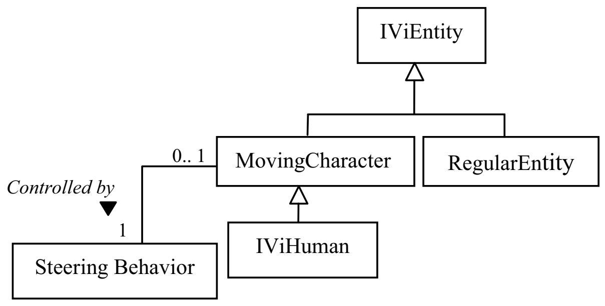


Figure 3- Class Diagram - IViEntity

The class IViEntity implements only an abstract representation of movement. It is not even connected with OGRE in any way other then by using its basic types (e.g. vectors). It serves as a base class that provides some basic movement functionality to other classes that inherit from it. Its functionality can be identified with part of the functionality of Reynolds' vehicle model and, once mapped into rendered motion, it is enough for entities whose movement is generated by some external force or will, that is, entities that do not need to appear as being self powered or as moving on their own.

MovingCharacter is one of the classes that inherit from IViEntity and it extends its movement functionality to the point that the vehicle model achieves, completing an abstract interface of the locomotion layer, over which the steering layer can be built. The MovingCharacter class also implements the features of the steering layer. It models a character that can move by producing forces to steer itself. These forces produce accelerations that are used to update the MovingCharacter's state. Steering behaviors determine forces that the character applies on itself, simulating how a self-powered entity uses energy to move by itself.

The vehicle model is used so that the steering layer can operate independently of the actual vehicle it is driving.

In the IViHumans platform, a MovingCharacter can have one instance of the class SteeringBehavior managing its motion (Figure 3). This steering behavior can either be a basic one or what we called a CombineBehavior, which is actually a composite [Gamma95, Larman98] steering behavior.

We implemented six basic steering behaviors: SeekBehavior, Seek2DBehavior, Arrive2DBehavior, Walk2DBehavior, Stop2DBehavior, and FollowPoints2DBehavior.

We decided to distinguish behaviors that operate in three dimensions from those that operate in two, by including the term “2D" in their name. Behaviors that operate in 3D are intended for characters that can move freely in the three dimensions of an environment (e.g. birds that can fly in the sky; fish that can swim in the sea). On the other hand, characters whose movement is constrained to one plane need behaviors that operate in 2D.

Seek2DBehavior is a particularization of SeekBehavior that functions only in two dimensions. The movement that this behavior generates is restricted to the horizontal plane that stands at the same height as the character. As the previous one, Arrive2DBehavior operates on the horizontal plane of the character. For most of the time, this behavior does the same thing as Seek2DBehavior but, when the character approaches the target, the computation differs. So, we had Arrive2DBehavior as a subclass of Seek2DBehavior, in order to reuse the implementation of this last one. It also drives the character in the direction of the target, but it makes him slow down linearly as he approaches it: the deceleration is applied only until the character's speed reaches some predetermined value, after what the speed is maintained. When the character comes very close to the target, as defined by another threshold, the behavior stops the character's movement and deactivates itself. In our opinion, this version produces more realistic results, in what concerns the movement of VHs.

We also implemented a steering behavior that drives the character at a given velocity and we called it Walk2DBehavior. This behavior has a target, like the previous ones, although this target is a velocity vector instead of a position. In each iteration, it calculates a force that will accelerate the character towards a velocity that can be obtained from the supplied target.

A steering behavior to stop the character-Stop2DBehavior- was also created. This behavior is a particularization of Walk2DBehavior, but the target velocity is defined a priori as the null vector. When the character stops moving, that is, when his velocity drops bellow a very low threshold, the behavior deactivates and removes itself from the MovingCharacter.

The last steering behavior we implemented was FollowPoints2DBehavior. It will drive the character across a sequence of n target positions and it will stop him once he comes to the last one. It does so by making the character seek the first n-1 targets and arrive at the nth target. In fact, it aggregates instances of Seek2DBehavior and Arrive2DBehavior to accomplish that goal, but that is completely transparent to the user.

Besides these basic steering behaviors, many more may be implemented in the future, simply by extending the abstract class SteeringBehavior.

4.3 Facial Expressions

In the IViHumans platform, VHs convey emotions through facial expressions. Any number of basic expressions may be modeled for a virtual human as deformations of the original model. These can be blended together to create composite expressions. As far as a VH is concerned, the distinction between basic and composite expressions is abstracted, so that he can uniformly exhibit and exchange them whenever needed. For the transition between two expressions to happen smoothly, their state is regulated according to a Finite State Machine (FSM) that specifies how their intensities vary in each time step. This FSM also establishes how the VH must deal with the expressions in specific circumstances (for instance, telling him to release an expression whose role has been fulfilled).

Basic expressions are wrapped in animations that last 1s and that are composed by two key-frames each: one at the instant 0s, that records no change in the 3D model, and one at the instant 1s, that records a shape of the model that corresponds to the expression in its maximum intensity. To exhibit different intensities of an expression, the animation is set at the corresponding instant. When more than one expression animation is manipulated this way, the VH exhibits a composite expression.

A VH can have only one main expression (though it can be composed). This expression can be activating – increasing its intensity until the desired value – or active – already with the intended intensity. Once the VH is told to show a new expression, this becomes the main expression and replaces the former which is put in a buffer of deactivating expressions. More than one expression can be deactivating for any VH. An expression will remain deactivating until it reaches zero intensity. If a new expression is provided before full activation of the current one is reached, the VH will handle it robustly, buffering several deactivating expressions while activating one, even if this happens repeatedly and successively.

Otherwise, and as long as transitions last a fixed amount of time, the main expression will always reach full activation at the same time that the previous expression finishes deactivating.

5. AN EXPERIMENTATION SCENARIO

In order to test the implemented features of the platform we have built an experimentation scenario representing an office open space, where several virtual humans, men and women, go about doing their usual tasks at their desks until they notice the occurrence of a fire in the office. At that point they must try to escape the office as soon as possible. Figure 4 is a snapshot of that experimentation scenario, showing 3 fire spots.

The office environment was modeled on Blender[4] and the furniture is composed by models available from Google SketchUp[5]. The fire spots were created using an add-on from Ogre 3D called Particle System[6] and they were wrapped on transparent bounding boxes, which can be detected by the vision algorithm.

[4] http://www.blender.org/

[5] http://sketchup.google.com/

[6] http://www.game-cat.com/ogre/pe/ParticleEditor_Beta.zip

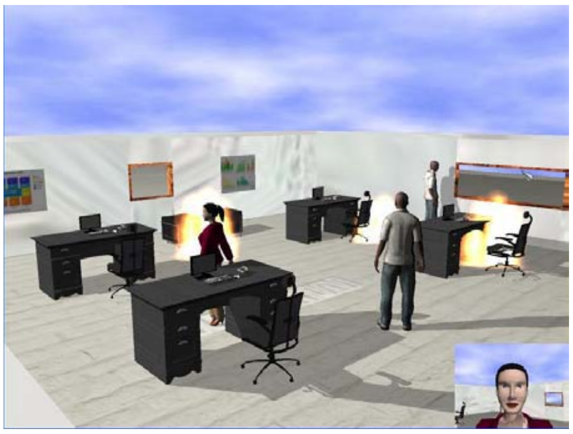


Figure 4 - Experimentation scenario

The office is inhabited by VHs of two kinds. The following paragraphs describe their particular characteristics and both the modeling and animation processes.

The creation of the female VH began with face modeling and with the definition of the six basic expressions identified by Paul Ekman [Abreu07a]. The face was modeled, using Blender, as a polygon mesh and the deformations that correspond to the basic expressions were codified as poses of this mesh. A pose records a set of displacement vectors for the vertices of the mesh. When the vertices are translated by the corresponding displacement vectors that a given pose specifies, the geometry of the mesh changes so that the intended expression is exhibited. If one intends to show an expression just partially, it suffices to reduce the length of the vectors that specify the translation of each vertex. On the other hand, the visualization of complex expressions can also be easily achieved by mixing basic expressions with desired intensities.

A tool was created for obtaining complex expressions by mixing basic ones, observing the effect in real-time. This tool was called Faces [Abreu07a]. The user can vary the intensity of each basic expression intuitively by playing with scroll buttons, and the results are immediately shown as a deformation of the face and new expressions can be saved. This way, libraries of facial expressions can be created for a model.

The body, cloth, and hair of the female VH were derived from a model we obtained in Poser[7] as a polygon mesh. The geometries of the meshes (body, cloth, hair, and face) were adjusted so that they would fit together and could be merged into a single continuous mesh [Abreu07a] [Abreu07b].

We prototyped only one animation for our virtual woman: a walk animation. To produce it in Poser, we used a very common technique: skeletal animation. We used the skeleton that was predefined for the chosen model and we defined its motion by parameterizing a base movement, relying on a set of features provided by

[7] http://www.e-frontier.com/go/poser

Poser. The skeleton, already imbued with the walk animation, was imported into Blender and associated to the mesh. This process involved consecutive adjustments of the mesh, to eliminate any residual inter-penetrations among body and clothes and among different parts of the body. The skeleton and the mesh were then exported to OGRE formats and went through a series of additional procedures that we conceived to overcome other obstacles [Abreu07].

The male VH was obtained from aXYZ[8] design and its materials are essentially made up of color maps that give it much more realistic appearance because they are properly done and include such details as cloth folds and wrinkles or beard. The association between mesh and skeleton proved to be quite perfect and we could easily create several poses without incurring the issues we had with the female prototype, which we have previously produced from scratch. We downloaded the model in Autodesk 3ds Max[9] format and readied it in a trial version of this software. We imbued the model with three animations: an idle animation (to use when the VH has to stand in the same place), a walk animation, and a run animation. These animations were derived from motion capture data we got from www.mocapdata.com. We obtained them in a 3ds Max proprietary format that could directly be applied to the skeleton of the VH- the biped format.

In both VH prototypes, the animations are composed by a set of keyframes that record successive postures of the skeleton. Each keyframe encodes a set of transformations that, when applied to the bones, put the skeleton in the corresponding pose. The keyframes are assigned unique instants of the animation, normally separated by regular time intervals. The posture of the skeleton in any animation is then generated by linear interpolation of the two closest keyframes, chronologically speaking (one that precedes it and another that comes after it). Each animation is created in such a way that it would remain coherent if its last frame was placed in the beginning of the animation and the rest were shifted. So, the animation can be repeatedly played with the first frame perfectly fit to succeed the last one. That is, the animations are cyclic and can be played unlimitedly.

6. THE AI LAYER

As mentioned previously, to allow the control of each virtual human to be as flexible as possible, software agents that can be programmed to incorporate as much complex and intelligent behaviour as needed are assigned the responsibility of controlling the actions of the VHs. So, the AI layer of the IViHumans platform corresponds to a multiagent system including not only the agents that directly control the virtual characters, the so-called cognitive agents, but also other types of agents.

[8]http://www.axyz-design.com/

[9]http://usa.autodesk.com/adsk/servlet/index?id=5659302&siteID=123112

The AI layer is being implemented using the JADE multi-agent system development platform.

For the moment, three types of agents have been implemented: MAS Porter, Interface Agents and Cognitive Agents. MAS Porter is a simple kind of agent that acts as a gateway to the CG layer server. It is this agent that forwards the requests from all the other agents of the AI layer to the graphical server and that retransmits back the received answers.

Interface agents are responsible for mediating between the cognitive agents and the CG layer server. Since cognitive agents can be conceived using different knowledge representations models and communication protocols, interface agents specific to each subtype of cognitive agent may have to be developed in order to allow them to interact correctly with the CG layer, as shown in Figure 5.

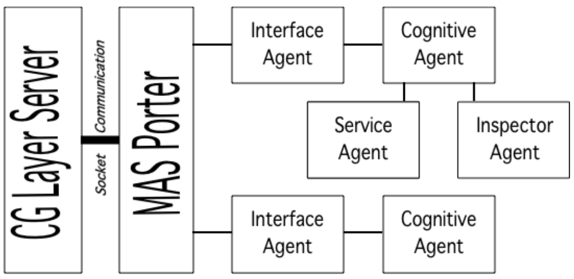


Figure 5 – AI Layer organization

Service agents, as the name suggests, provide extra services to the community. They can be shared by various cognitive agents, providing common facilities, available to all, instead of having each agent support its own version of the commodity. Service agents can, for instance, provide a path planner service, from which an agent can request a sequence of actions to reach a desired location, or external services providing agents with links to outside sources, namely input from a web page.

Other types of agents have been conceived, but have yet to be implemented. One such type of agent is an introspection agent that obtains information from the cognitive agents about their internal state, for instance their emotional state or their achievement goals, and adapts the data for external visualization. Other types of special agents that help control simulations are demonstrators (that can conduct online demonstrations of the execution of a certain task, and are able to modify task's starting situation and parameters in order to illustrate alternative courses of execution) and testers (that can conduct large sets of demonstrations, in different setups, as a way to test the correctness or the performance of certain task procedure).

In the following subsections, we will present some additional details about the cognitive and interface agents.

6.1 Cognitive Agents

The IViHumans platform does not impose a specific internal model that the cognitive agents must conform to. For instance, they may be conceived as BDI agents, or as reactive agents, or follow any other agent model that may seem appropriate to control a virtual character. Almost the only requirement is that any cognitive agent must first assure the assistance of an appropriate interface agent, able to convert his requests to those accepted by the CG layer server in order to control a specific VH.

This allows the reuse of intelligent agents that might have been developed previously, with small changes to allow them to control a VH in the platform, with the help of a specific interface agent.

Until now, we have implemented two types of cognitive agents that illustrate different development approaches. Both were used in the experimentation scenario of figure 4.

One type of cognitive agent that we implemented was specifically developed for this platform and uses internal representations of what it senses (the virtual objects he sees, whose identification and properties he must request from the CG layer) that are identical to those transmitted by the graphical layer. The atomic actions it considers while planning the actions of the VH it controls correspond exactly to the steering behaviours that the graphical layer server is able to activate. It therefore needs no specific translation services from an interface agent. It was conceived as a BDI agent and implemented using JADEX[10]. It does not however model emotions, so it is being used in the experimentation scenario to control the virtual man, whose animations do not yet include the capability of expressing facial emotions.

The other type of cognitive agent that we implemented, called EmoCognitiveAgent, is able to express emotions and so it was modelled based on the Agent Flow Model [Morgado08], reusing an already implemented agent developed in the context of another project, called AutoFocus, and whose internal representations of sensory information and simple actions differ significantly from those proper to the IViHuman CG layer server. Also, while in the IViHuman platform it is usually assumed that sensors are passive and that cognitive agents must take the initiative of requesting sensory information when needed, in the AutoFocus [Neves09] platform where this EmoCognitiveAgent was originally developed the approach was the opposite: It is the environment that takes the initiative of sending the agent its present observed state and expects to receive back, as an answer, the action to be performed by the agent and the emotional expression to display. Therefore, in the case of the EmoCognitiveAgent there was a need to develop a dedicated type of interface agent that could initiate and take control of the communication with the CG layer server, but also that could translate the sensory information, the actions to be performed and the emotions to be expressed between the two representations.

[10] http://jadex.informatik.uni-hamburg.de/

In the next subsection, we present a general description of the type of functions that interface agents can perform. Subtypes of interface agents are being defined and prototype implementations developed, to serve the needs of particular types of cognitive agents.

6.2 Interface Agents

Interface agents can act as a raw connection between both layers, but they can also have two additional functions: to provide a sensing/acting cycle that further separates the communication aspects of the control of the VHs from the more complex, and possibly slower, cognitive aspects; and to offer a translation/filtering mechanism between crude data and cognitive agents' adopted representations.

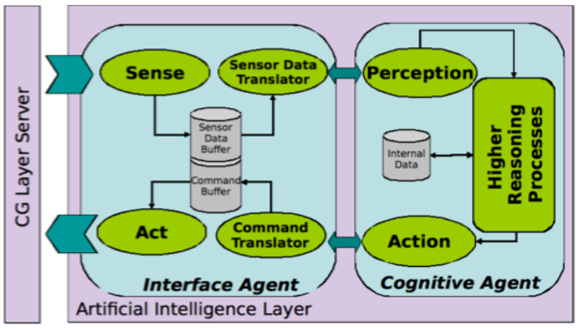


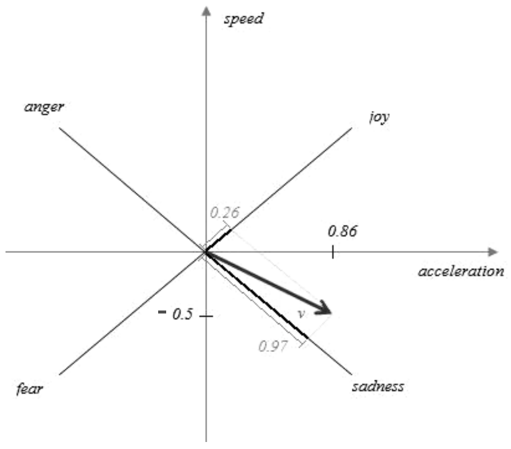
Figure 6 – Interface and cognitive agents internal components

Accordingly, as depicted in Figure 6, interface agents can be decomposed into four main components:

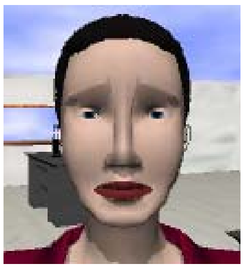
- Sense: requests sensory information from the GP layer at a defined rate, saving it into a buffer (the sensor data buffer). This saves the higher cognitive levels from having to deal with the sensor particulars (refresh rate and cycling).
- Sensor data translator/filter: translates raw sensor information into symbolic equivalents or more abstract and constrained representations. Clustering of similar data or aggregation of correlated data may be involved in this translation process. For instance, a color name can be made to correspond to an interval of values in the rgb gamma or several distinct observed objects can be merged into a single bigger object.
- Command translator: translates the higher level commands used by the cognitive agent into the kind of commands accepted by the GP layer (and saves them in the command buffer). The translation can be achieved by using predefined schemas. Another alternative is to incorporate a planner that produces in real time the desired low level command sequence.
- Act: reads the next command from the command buffer and sends it to the GP layer. This feature detaches the cognitive agent from the physical details

such as the number of commands that the GP layer is capable of processing in a time slot.

As a specific case of command translation, we will describe the translation of emotional information transmitted by the EmoCognitiveAgents above referred into request for showing a certain facial expression of the woman virtual character. The EmoCognitiveAgents follow the Agent Flow Model where emotions are represented by so called emotional dispositions. These are not discrete emotional labels but rather are represented as two-dimensional numerical vectors that continuously change, derived from the speed and acceleration with which the agent thinks it is progressing towards its own goals. Those vectors must then be converted into one of the six basic emotional facial expressions that the virtual woman in the platform is able to present, or a composition of such basic expressions. The approach taken to perform that conversion was based on the fact that already in the Agent Flow Model the quadrant of the two-dimensional space where an emotional disposition vector lays can be used to assign it a discrete label, that very roughly represents its main emotional tendency: joy, anger, fear or sadness, for vectors laying respectively in the first to fourth quadrants.



(a)



(b)

Figure 7: From internal emotional dispositions to facial expressions

Following that idea, an emotional disposition vector laying over the bisector vector of one of the quadrants is translated into the basic emotion associated to that quadrant, and any other emotional disposition is translated into a composition of two basic emotions, the ones associated to the quadrant bisector vectors closer to that emotional disposition vector. The intensities of the two elements of that composition correspond to the correlation between the bisector vector and the emotional disposition vector. In Figure 7, part (a) illustrates the translation of the emotional disposition vector ***v*** into a composition of sadness, with intensity 0.97, and joy, with intensity 0.26, and part (b) shows the facial expression corresponding to that translation.

7. CONCLUSIONS

Our goal is to develop a general graphical visualization platform for multi-agent system execution and to apply it to the development of realistic and compelling simulation environments, and to incorporate results obtained in the area of emotion modeling. We believe that the platform IViHumans may become a valuable tool for training and simulation based design purposes.

Recent developments in the AI layer have enabled us to build a demo, with embodied agents, that explores the connection between the two layers. Namely, it allows AI agents to use the services provided by the GP layer. This demo illustrates that the VHs are capable of perceiving the environment and, according to this information, they are able to decide and react.

In the near future it is important to include more models and scenarios. Besides the animations the models currently have, several more would be desirable, to enable actions like sitting, grasping, and moving the head. Additionally, an IK solver could be included, either to deviate existing poses or to create new ones on the fly.

8. REFERENCES

[Abreu08] Ricardo Abreu, Ana Paula Cláudio, Maria Beatriz Carmo, Luís Moniz, Graça Gaspar. Virtual Humans in the IViHumans Platform. In Proc. of 3IA 2008, International Conference on Computer Graphics and Artificial Intelligence, in cooperation with Eurographics, pages 157-162, Athens, Greece, May 2008.

[Abreu07a] Ricardo Abreu, Ana Paula Cláudio, Maria Beatriz Carmo. Humanos Virtuais na Plataforma IViHumans: a Odisseia da Integração de Ferramentas. In Actas do 15º Encontro Português de Computação Gráfica, 15º EPCG, pages 217-222, October 2007.

[Abreu07b] Ricardo Abreu, Ana Paula Cláudio, Maria Beatriz Carmo. Desenvolvimento de Humanos Virtuais para a Plataforma IViHumans. Technical Report DI-FCUL TR-07-32, Departamento de Informática da Faculdade de Ciências da Universidade de Lisboa, November 2007.

[Barella06] A. Barella, C. Carrascosa, and V. J. Botti. JGOMAS: game-oriented multi-agent system based on JADE. In Adv. in Comp. Entertainment Technology. ACM, 2006.

[Buckland05] Mat Buckland. Programming Game AI by Example. Wordware Game Developer's Library. Wordware Publishing, 2005.

[Evertsz07] R. Evertsz, F. E. Ritter, S. Russell, and D. Shepherdson. Modeling rules of engagement in computer-generated forces. In Proc. of the 16th Conf. on Behavior Representation in Modeling and Simulation, pages 123-134, 2007.

[Funge99] John Funge, Xiaoyuan Tu, and Demetri Terzopoulos. Cognitive Modeling: Knowledge, Reasoning and Planning for Intelligent Characters. In Siggraph 1999, Computer Graphics Proceedings, pages 29-38, Los Angeles, 1999. Addison Wesley Longman.

[Larman98] Graig Larman. Applying UML and Patterns. An Introduction to Object-Oriented Analysis and Design. Prentice-Hall, 1998.

[Longhi04] Magalí Longhi, Luciana Nedel, Rosa Viccari, and Margarete Axt. Especificação e Interpretação de Gestos Faciais em um Agente Inteligente e Comunicativo. In SBC Symposium on Virtual Reality, São Paulo, 2004.

[Magerko04] Brian Magerko, John E. Laird, Mazin Assanie, Alex Kerfoot, and Devvan Stokes. AI Characters and Directors for Interactive Computer Games. In 16th Innovative Applications of Artificial Intelligence Conference, pages 877-883, 2004.

[Morgado07] Luís Morgado and Graça Gaspar. Abstraction Level Regulation of Cognitive Processing Through Emotion-Based Attention Mechanisms. In Lucas Paletta and Erich Rome, editors, WAPCV, volume 4840 of Lecture Notes in Computer Science, pages 59-74. Springer, 2007.

[Morgado08] Luís Morgado, Graça Gaspar, Towards Background Emotion Modeling for Embodied Virtual Agents, Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008), Padgham, Parkes, Mueller and Parsons (eds.), Portugal, pp. 175-182, May 2008

[Neves09] Pedro Neves, Graça Gaspar, Luís Morgado, AutoFocus Framework Documentation, RT-DITR-09- 1, January 2009

[Norling01] Emma Norling and Frank E. Ritter. Embodying the JACK Agent Architecture. In Brooks M., Corbett D., and Stumptner M., editors, AI 2001: Advances in Artificial Intelligence, volume 2256 of LNCS, pages 368-377, 2001.

[Norling04] Emma Norling and Liz Sonenberg. Creating Interactive Characters with BDI Agents. In Australian Workshop on Interactive Entertainment, 2004.

[Noser95] Hansrudi Noser, Olivier Renault, Daniel Thalmann, and Nadia Magnenat Thalmann. Navigation for Digital Actors based on Synthetic Vision, Memory, and Learning. Computers and Graphics,19(1):7-19, 1995.

[Perlin96] K. Perlin, A. Goldberg. Improv: A System for Scripting Interactive Actors in Virtual Worlds. Computer Graphics, 30 (Annual Conference Series): 205-216, 1996.

[Peters02] C. Peters, C. O Sullivan. Synthetic vision and memory for autonomous virtual humans. In Computer Graphics Forum, volume 21, pages 743-752. Blackwell Publishing, November 2002.

[Reynolds99] Craig W. Reynolds. Steering Behaviors for Autonomous Characters. In Game Developers Conference 1999, 1999.

[Reynolds87] Craig W. Reynolds. Flocks, Herds, and Schools: A Distributed Behavioral Model. Computer Graphics, 21(4):25-34, 1987.

[Rickel97] J. Rickel and W. Lewis Johnson. Integrating Pedagogical Capabilities in a Virtual Environment Agent. In Proc. of the First Int. Conf. on Autonomous Agents (Agents'97), pages 30-38, New York, 1997. ACM Press.

[Schmidt00] Douglas Schmidt, Michael Stal, Hans Rohnert, and Frank Buschmann. Pattern-Oriented Software Architecture, Volume 2, Patterns for Concurrent and Networked Objects. John Wiley & Sons, 2000.

[Semião06b]Pedro Miguel Semião, Maria Beatriz Carmo, and Ana Paula Cláudio. Algoritmo de Visão para Humanos Virtuais. In Actas da 2a Conferência Nacional em Interacção Pessoa-Máquina, Interacção 2006, pages 133-138, Outubro 2006.

[Semião06a] Pedro Miguel Semião, Maria Beatriz Carmo, and Ana Paula Cláudio. Implementing Vision in the IViHumans Platform. In Ibero-American Symposium on Computer Graphics, SIACG 2006, pages 56-59, July 2006.

[Torres03] J. A. Torres, L. P. Nedel, and R. H. Bordini. Using the BDI Architecture to Produce Autonomous Characters in Virtual Worlds. In IVA, pages 197-201. Springer, 2003.

[Torres04] J. A. Torres, L. P. Nedel, and R. H. Bordini. Autonomous Agents with Multiple Foci of Attention in Virtual Environments. In Int. Conf. on Computer Animation and Social Agents, pages 197-201, 2004.

[Ulicny01] B. Ulicny, D. Thalmann. Crowd Simulation for Interactive Virtual Environments and VR Training Systems. In Eurographics Workshop of Computer Animation and Simulation '01, pages 163{170. Springer-Verlag, 2001.

[Vosinakis01] Spyros Vosinakis and Themis Panayiotopoulos. SimHuman: A platform for Real-Time Virtual Agents with Planning Capabilities. In IVA 2001 3rd InternationalWorkshop on Intelligent Virtual Agents, pages 210-223. SpringerVerlag, 2001.