GeDA-3D a Middleware Useful to Handle the Evolution in *Behavioral Animation*-Based Virtual Worlds with a Multi-agent Architecture

Félix F. Ramos, H. Iván Piza, and Fabiel Zúñiga

Multi-Agent Systems Development Group at CINVESTAV del IPN, Guadalajara, Jal., México
{framos,hpiza,fzuniga}@ gdl.cinvestav.mx
http://gdl.cinvestav.mx

Abstract. In this article is proposed a distributed middleware useful to handle the evolution of deterministic virtual scenes in a 3D world. The proposed middleware allows interaction necessitated among virtual humans [1] and the environment. This interaction allows virtual humans to get user defined goals. As stated in the behavioral animation [3,4] paradigm, the user only tells characters "what to do" (goals) instead of "how to do it" (actions). Every virtual human computes dynamically by means of an intelligent algorithm, the actions to achieve its goal based on: a) its actual state; b) the stimuli perceived from the environment, and c) the personality of the virtual human. Main components of the proposed middleware is part of a more complex system we call GeDA-3D [5,6], this system includes a Declarative Virtual Editor useful to create the virtual world, an Context Descriptor used to define the physic laws ruling the environment, increment the language declarative language with definitions, concepts etc. A Rendering Tool useful to display the evolution of the scene, an Agent's Control module to control the agents managing the different virtual life creatures and all this is around a Geda-3D's kernel that provides all the stuff necessary to all these modules interact. Briefly the behavior of these middleware is: The scene controller receives the actions, validates them, handles the effect of the actions according to the natural laws of the world, resolves a set of graphic primitives to render and launches an event for every goal achieved. The cycle of sending local states and receiving actions loops until no goal is left to fulfill.

1 Introduction

The development of distributed systems is well adapted to human behavior; however it is necessitated at same time to offer better interfaces to humans. With this aim, we propose a middleware useful to develop distributed application where a 3D interface is useful. More of these distributed applications recreate a virtual environment o virtual world. When we say a virtual world we mean all programs necessitated to represent a world with all things objects creatures interactions etc. For instance it a world can be all the stuff necessitated in a manufacture plant or in a thermoelectric station. The term virtual creature is typically defined as a representation of a real or fiction creature generated and managed by a computer. Virtual creatures have gained great popularity in the recent years, especially on the entertainment industry – to develop animated movies and video games- and on the academic field, to recreate prehistoric

T. Böhme et al. (Eds.): IICS 2004, LNCS 3473, pp. 168–177, 2006.

environments. When dealing with real-time applications, for instance video-games, it is not so simple to handle the behavior of the virtual humans to make them look realistic.

As a consequence, several researchers from around the world have led their works toward the behavioral animation [3,4] of virtual humans in order to create virtual scenes less predictable, where virtual humans take autonomous decisions in real-time. The spirit of behavioral animation techniques lays in the concept of telling characters "what to do" instead of "how to do it". In behavioral animation, a virtual human determines its own actions, at least to a certain degree. This gives the virtual human an ability to improvise, and frees the animator from the need to specify each details of every human's motion. Unfortunately, it is not clear how the current works provide means for users to: a) develop behaviors and assign them to virtual humans, b) specify the natural laws ruling the interactions in a virtual world, and c) take advantage of existing rendering tools in order to avoid low-level implementations.

We propose an agent-oriented middleware useful to handle the evolution of virtual scenes based on behavioral animation. This middleware is the core of a complex system called GeDA-3D, and introduced in [5,6]. The main goal of this first stage of GeDA-3D concerns allowing inexpert users –scenarists, hereafter– to provide a human-like description of any scene and, as a result, a 3D-graphical representation of such scene is rendered. This overall operation is depicted in Figure 1. During the evolution of the scene, virtual humans [1,2], or characters, interact with each other in order to achieve a number of goals specified formerly by the scenarist.

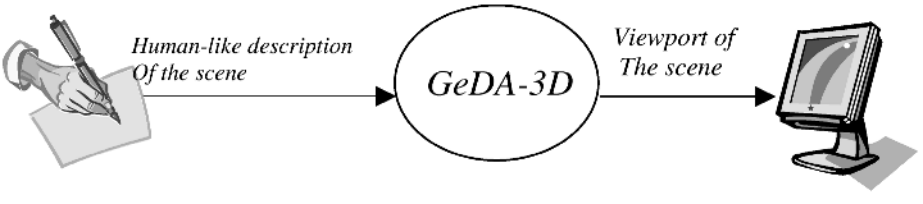

Fig. 1. Context-DFD of GeDA-3D

This article is organized as follows. Section 2 introduces the architecture of the system together with a brief explanation of the components' competencies. Section 3 depicts the internal operation of an agent in GeDA-3D. Section 4 shows the information flow between the components of the middleware. Section 5 summarizes the conclusions and future work.

2 Architecture

The middleware proposed in this paper follows an agent-oriented architecture which allows management of distributed applications and assists the development of virtual environments. It represents the core of GeDA-3D. The behaviors of the characters are actually distributed applications treated as agents attached to the platform. An agent can control the behavior of more than one character. The platform provides templates allowing the programmer to easily develop simple or complex behaviors.

The core of GeDA-3D is primarily in charge of: a) sending the agents the next set of goals their characters have to fulfill and the current local state of the world, b) managing the interaction of the virtual entities according to the natural laws of the world being modeled, and c) sends a set of lower-level commands to a 3D-tool to render the scene. Every agent sends back to the core an action sequence to encourage the fulfillment of the current goal. Figure 2 depicts a data-flow diagram of GeDA-3D considering some components required to accomplish the main goal of the project.

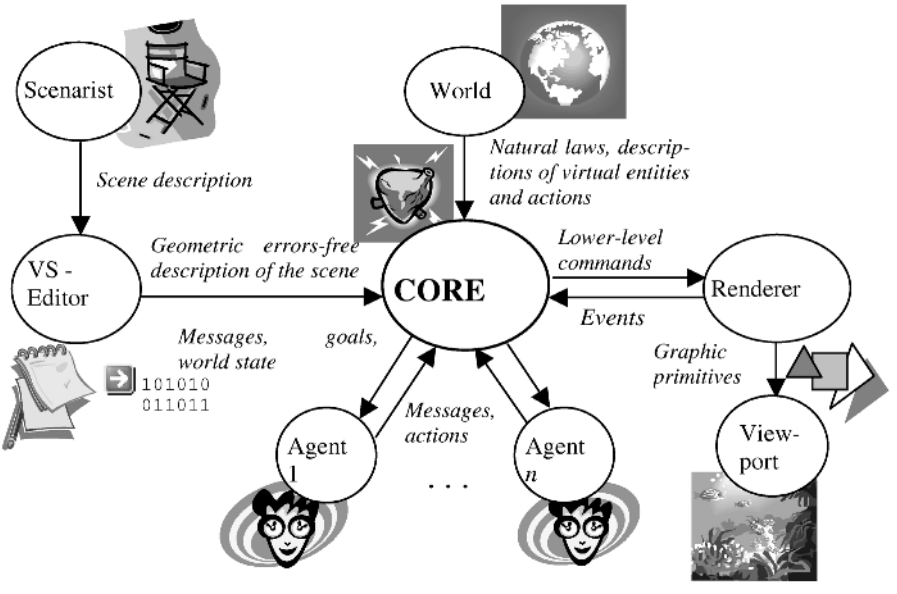

Fig. 2. Level 0-DFD of GeDA-3D

The overall operation of GeDA-3D is as follows. The scenarist provides a declarative [7] description of the scene using geometric constraints [8]. Such description includes: a) creation of virtual entities, b) assignment of behavior for characters, c) arrangement of entities in the world, and d) description of the goals to be fulfilled by the characters. The Virtual Scene (VS) Editor translates the declarative description in a geometric one, only if the description is errors-free and valid according to a number of rules. The world (or context) defines a set of natural laws that rules the interaction between the virtual entities, and includes descriptions of the actions and entities admissible.

The core manages the evolution of the scene: it validates and executes actions, and handles the effect of interactions between entities. Every agent receives the state of the world and sends back actions to achieve the goal-specification. Finally, the renderer receives low-level commands useful to display continuously the scene.

The architecture of the core is constituted primarily by two modules called *scene* and *agents control*. The scene control addresses all the issues related to the evolution of the scene as a result of the interaction of characters, while the agents control manages the distributed issues of the platform. Figure 3 shows a more detailed data-flow diagram depicting the architecture of the middleware proposed.

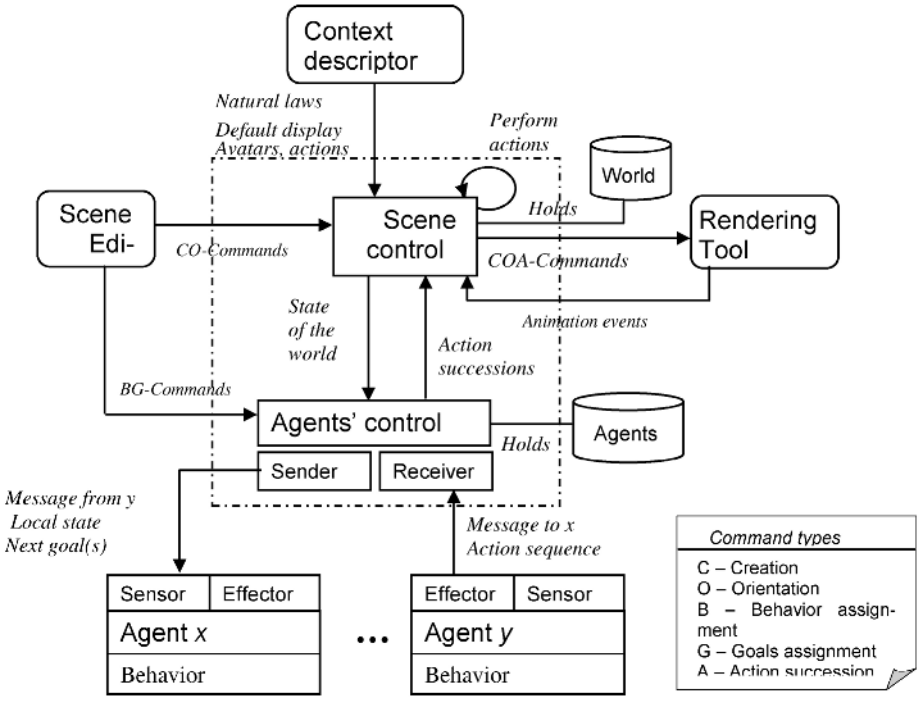

Fig. 3. Middleware architecture

2.1 Scene Control

The scene control is in charge of control the interactions and evolution of the virtual scene. It receives high-level commands to, initially, create and arrange virtual entities in the scene and, iteratively, to request the execution of actions. As a result, the scene control produces low-level commands to be sent to a specific renderer who translates each command in a set of graphical primitives. A low-level command indicates either the creation/placement of a virtual entity, or the animation of one or more entities. A high-level command can produce zero or more low-level ones. The tasks performed by the scene control include the following:

- Receive a sequence of high-level commands for the display of the virtual objects in a certain position
- Receive periodically – coming from the agents – sequences of actions for the characters to perform
- Validate the execution of all the actions received; it has the ability to delay or cancel the execution of a single action or an action succession
- Resolve a sequence of animations to execute as a result of an interaction, and send it to the renderer; the natural laws have much influence in this task
- Whenever an action succession is finished, successfully or prematurely, the scene control determines and sends updated local world-states to the agents through the agents' control. A premature finalization is usually due to unexpected contacts

with other moving entities or by gravity effects: the ground was not enough for all the actions. In either case, the agents are notified right away with the new local world-state.
- It holds a World database containing fundamental information about all the virtual entities involved in the scene

2.2 Agents Control

The agents controlling the behavior of characters are actually distributed applications. The agents' control manages the connection and disconnection of the agents and all the issues related to the transference of data between the community of agents and the middleware. Such data include the following:

- Control assignment. Every agent controls the behavior of one or more characters and enforces them to accomplish a goal-specification defined by the scenarist. A character is said to be linked to an agent during the scene. More details about the agent operation can be found in Section 3.
- Goals required. The operation of an agent is oriented to achieve one or more goals not yet fulfilled and already enabled. A goal is enabled as soon as some preconditions are given, including external events or previously-assigned goals fulfilled. Such conditions are defined during the scene description.
- Local state of the world. Every character has a particular perception of the world around. This perception depends on the location and orientation of the character, and is represented as a set of entities reachable from the character's field of view. The information of an entity seen can be full or partial depending on how far it is from the eyes of the character.
- Action sequences. An agent computes an action sequence as a result of the current goal required, the personality definition and the local perception. The agents control continuously receives action sequences from all the agents and sends them to the scene control for validation and execution.
- Messages between agents. Very often, characters work as a team and hence need cooperation from each other, for instance, to increase the local perception of the environment. In such cases, agents are capable to exchange messages in order to provide information about the scene.

3 Agent Architecture

This section introduces a generic architecture useful to develop the behaviors of the characters. These behaviors are attached to the platform as distributed agents. The scenarist specifies *what* characters must do, instead of *how* they have to do it. The behavior is in charge of solving the second part. Therefore, two similar specifications might produce different simulations. The agent relies on a Beliefs, Desires and Intentions architecture, as described by Georgeff [9], so that each agent makes decisions by itself. In general terms, the agent operation consists on iteratively receiving goals and current states of the environment, and sending back action sequences. The local perceptions received from the environment represent agent's beliefs. Before introducing the agent architecture, some background concepts must be defined.

A **skill** symbolizes a non-trivial action or complex action [10,11], that a character knows how to carry out, and involves the execution of an appropriate sequence of *primitive* (trivial) actions. In order to start working on a skill, some initial conditions about the current status (location, visual state, orientation) of the virtual entities involved have to be fulfilled. Likewise, there are similar final conditions indicating a skill was successfully carried out.

The **goals** symbolize the author's requirements and an agent's desires. All the interactions occurring in a scene lead to the fulfillment of the goals submitted by the scenarist. A simulation is finished as soon as all the goals have been achieved successfully. A goal specification includes a character, a skill and a target. The target is optional and represents either a zone in the environment, or a virtual entity. Sometimes, the scenarist structures the goal-description in such a way that sets of goals enable other sets of goals. That is, goals g_{j1}, g_{j2}, … g_{jn}, have to be achieved (concurrently, sequentially, or optionally) necessarily before start working with goals g_{k1}, g_{k2}, … g_{km}. In this context, we call *j* and *k* **goal-cycles**. Whenever a goal-cycle is started, an agent is told to fulfill a new set of goals.

Often, the fulfillment of a goal is not a straightforward process of just selecting a series of actions and that's all. Since we are dealing with dynamic environments, the actions selected at a specific time may not lead us to the goal, because of unexpected presence of new obstacles or adversary entities. Therefore, characters frequently divide their current goals in **tasks** or subgoals, where every task lists a sequence of actions required to solve the current trouble. The tasks are also useful to achieve the initial conditions of the skill involved in the current goal. The tasks represent the intentions of an agent.

A **personality** is defined as everything that makes a difference between the behavior of two different characters sharing the same set of goals, skills and actions. The personality resolves the best sequence of actions (within tasks) in order to achieve the current goal of the character, according to the current state of the character inside the environment.

Figure 4 depicts the architecture of an agent. *Msgs* stores all the messages received by an agent; the purpose for the message-passing is to allow cooperation between agents, in the cases where an agent has information useful for any other. *GNF* (Goals Not Fulfilled) stores the goals left to fulfill, while *GF* (Goals Fulfilled) stores the goals successfully achieved. Let $\mathcal{A}$, $\mathcal{N}$ be, respectively, the set of all the actions and entities valid in the current context where the scene carries out, and $\mathcal{G}$ be the set all the goals specified by the scenarist.

The task-tree depends on the current skill selected and it is built up from a Process Algebras [12] expression. The following operators are considered:

- $t_1 >> t_2$: Prefix operator: The agent will have to perform first task t_1, then task t_2
- $t_1 [] t_2$: Choice operator: The agent perform either task t_1 or t_2.
- $t_1 ||| t_2$: Parallel composition operator: The agent must perform goals t_1 and t_2 without concerning the order of execution.

GA is a Genetic Algorithm embedded in every agent, which computes the best sequence of actions to perform. The objective function (*OF*) in *GA* is based on: a) the coordinates of the target place the character is leading to through the execution of the current task, and b) the state of the world, i.e. the presence of moving entities avoid-

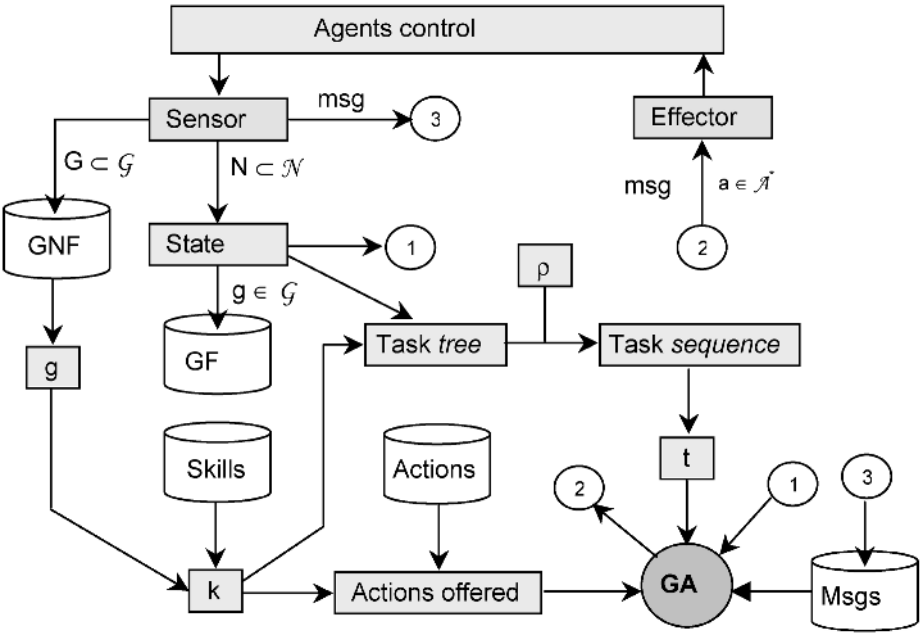

Fig. 4. Agent architecture

ing straight trajectories toward the target. A chromosome represents a sequence of actions to carry out. Two chromosomes may have different length. An allele is a pair <orientation, action> where orientation denotes the amount of degrees to turn about the 3-axes before carrying out the action. Two genetic operators are employed to get the offspring: crossover and mutation. The former combines two chromosomes using random crossover points. The mutation involves slight random changes in the orientation and action values, and the size of the chromosome. *OF* is in terms of: a) the distance between the character and the target, b) the trajectory length, and c) the presence of obstacles during the trajectory computed. The evaluation function computes large fitness values if the *OF* is minimized.

The agent operation is as follows:

1) At goal-cycle *n*, the sensor receives the next set of goals G to fulfill, and they become GNF; a goal *g* is selected from GNF. If GNF was not empty at cycle *n*, then the previous goals in GNF are discarded.
2) The underlying skill *k* from *g* is found in *Skills*, and a set of actions available for *k* is obtained
3) A task-tree structure is created according to the current state of the world and the current skill *k*.
4) The personality ρ chooses a task sequence from the tree.
5) Having as input data: a) the current local state of the world, b) a set of actions available, c) the current task and d) a list of messages, GA computes a sequence of actions and sends it to the effector; additionally, the GA may send messages to other agents through the effector.

6) Whenever the sensor receives a new local state of the scene, the agent checks if the current goal was successfully achieved; if positive, then a new goal *g* is obtained from GNF.

4 Information Flow

This section introduces the information flow between the VS-Editor, the core and the community of agents. Figure 5 depicts a zoom-in of the architecture in Figure 3, and shows a more detailed interaction between the agents control, the scene control and the community of agents. Such interaction involves sending and processing the components in a goal-description.

In addition, we introduce two modules, a goals manager and an event launcher, which belong to the agents control and the scene control, respectively. The former is in charge of: a) retaining the goals that are not enabled yet because of the lack of events, and b) sending back to the agents control those goals required to be fulfilled at the current time. Then, the agents control sends every agent the goals assigned to the characters it controls. The event launcher is given the state of the world every time it is changed, as a consequence of actions, and then it determines whether an event can be launched or not. This decision is taken using first-order logics. The behavior of all the components of the middleware is specified using process algebras notation, where they are treated as agents. First of all, let's define some functions and sets useful to resolve the set of goals an agent has to fulfill in a specific time, and hence, tu understand the information flow.

Let C be the set of characters (intelligent virtual entities) existing in the current scene, G be the set of goals assigned to characters, A be a set of agents available within a specific context, Σ is an indexing set of goal-cycles, and E be a set of events occurred in the scene. An event has one of two possible values: true or false. P stands for the power function.

Let $\delta_\sigma : C \rightarrow P(2^E \times G)$ be a function that assigns goals to a character, to be fulfilled during goal-cycle σ. Each goal will be enabled as soon as all the events associated are set to true. Let φ_σ be a logical expression that, if true, indicates the fulfillment of the required goals, and so, the finalization of a goal-cycle. This expression is built up with events. The logic operators employed are: $\wedge$, $\vee$, $\Rightarrow$. Δ, Φ denote, respectively, sets of δ and φ. More details about computing (Σ, E, G, Φ, Δ) from the goal-specification will be included in a further paper.

Let λ: $A \rightarrow 2^C$ be *a* mapping *from agents* to characters, and having the following constraint: two different agents are not allowed to control *the same character*. Formally, $\forall c \in C$, if $c \in \lambda(a_1)$, then $c \notin \lambda(a_2)$, having $a_1, a_2 \in A$ and $a_1 \neq a_2$.

Let FIL: $\mathcal{P}(2^E \times G) \rightarrow 2^G$ be a filtering function that resolves a set of goals enabled and not fulfilled yet, according to the events associated to each goal. Formally:

Let $\mathcal{D}$ stand for the domain of the FIL function, $e_1,\ldots,e_n \in E$, $g_k \in G$
$\forall <\{e_1, \ldots, e_n\}, g_k> \in \mathcal{D}$, if $v(e_1 \wedge \ldots \wedge e_n) = true$ and $v(e_k) = false$
then $g_k \in FIL(D)$.

Finally, let δ': $A \rightarrow \mathcal{P}(C \times 2^{\underline{G}})$ be a goal-assignment relation for agents ∋ $\forall a \in A$, if $c = \lambda(a)$, then $<c, FIL(\delta_\sigma(c))> \in \delta'(a)$, having $c \in C$, $\sigma \in \Sigma$. This relation tells an

agent which goals are currently enabled and required to be fulfilled by the different characters the agent controls.

AC $\stackrel{def}{=}$ $in_1(\Sigma, \Phi, \Delta, G, E).out_1(\Sigma, \Phi, \Delta).out_2(G)$ +
$in_2(\delta').out_i(\delta'(a_i))\ldots out_k(\delta'(a_k))$
+ $in_i(actions_i).out_3(actions)$ + $in_3(e).out_4(e)$

GM $\stackrel{def}{=}$ $(in_4(\Sigma, \Phi, \Delta) + in_5(e)).get\delta'().out_5(\delta')$

getδ'() $\stackrel{def}{=}$ if $v(\varphi_\sigma)$ is *true* then $\sigma = next\_\sigma()$
if σ is not *null* then compute δ' as follows:
$\forall a \in A, \forall c \in \lambda(a)$: $\delta'(a) = <c, FIL(\delta_\sigma(c))>$

SC $\stackrel{def}{=}$ $in_6(actions).perform(actions).out_6(world)$

EL $\stackrel{def}{=}$ $(in_7(G, E) + in_8(world)).event()$

event() $\stackrel{def}{=}$ if a new event e is launched then $out_7(e)$

AG_i $\stackrel{def}{=}$ $in_9(\delta'(a_i)).action().out_8(actions_i)$

action() $\stackrel{def}{=}$ run the underlying intelligent algorithm to compute the next set of actions for every character controlled

Notice that the agent control has a pair of ports $<in_1, out_i>$ (input and output) for every different $agent_i$ available in the current context. A further work will introduce the formal description of the actions assigned to characters, the virtual world and the event handling. The last one involves declaring a number of logic-based rules allowing an event to be launched.

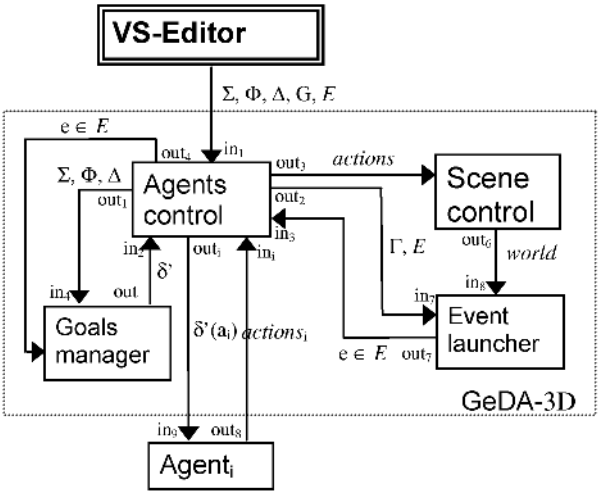

Fig. 5. Information-flow across the middleware

5 Conclusions and Future Work

In this paper we present our work regarding the development of a middleware useful to develop distributed applications where a 3D interface is useful. The design of this middleware is based on agents. Main characteristic of our middleware is that it hides from final user. Till now our middleware allows interactions completely specified. This is useful for applications where user specify all interactions goals to be achieved.

For instance, in entertainment industry this is useful to in the development of a theater scene or a chapter of a film. In industry, the construction of a product must be completely specified. We have work with some very simple examples for instance the simulation of a soccer game, a small piece of theater.

Our future work is addressed in two directions first to provide formal specification of our middleware and second more applicative: to improve our middleware to support interactions not specified; to allow user interact with applications, for instance allowing the control of objects; to evolve our character agent architecture to support in case of virtual human some characteristics like mood, and personality. Also we are developing infrastructure to support several real time engines for rendering the 3D results. In parallel we are developing some applications for final users.

References

1. Badler, N. Real-Time Virtual Humans. Pacific Graphics (1997)
2. Musse, S., Garat, F., Thalmann, D. Guiding and Interacting with Virtual Crowds in Real-Time. In Proceedings of EUROGRAPHICS Workshop on Animation and Simulation. Milan, Italy (1999) 23-34
3. Thalmann, D. Monzani, J.S. Behavioural Animation of Virtual Humans: What Kind of Laws and Rules? In Proc. Computer Animation 2002, IEEE Computer Society Press (2002) 154-163
4. Reynolds, C.W. Flocks, herds, and schools: A distributed behavioral model. Proceedings of SIGGRAPH 87 (1987) 25–34
5. Ramos, F., Zúñiga, F. Piza, I. A 3D-Space Platform for Distributed Applications Management. International Symposium and School on Advanced Distributed Systems 2002. Guadalajara, México. ISBN 970-27-0358-1 (2002)
6. Piza, I., Zúñiga, F., Ramos, F. A Platform to Design and Run Dynamic Virtual Environments. CyberWorlds. Tokyo, Japan. ISBN 0-76952140-1 (2004) 78-85
7. Kwaiter, G., Gaildrat, V., Caubet, R. DEM²ONS: A High Level Declarative Modeler for 3D Graphics Applications. In Proceedings of the International Conference on Imaging Science Systems and Technology, CISST'97. Las Vegas (1997) 149-154
8. Le Roux O., Gaildrat V., Caubet R. Design of a New Constraint Solver for 3D Declarative Modeling: JACADI. 3IA: Infographie Interactive et Intelligence Artificielle, Limoges (2000)
9. Rao, A.S., Georgeff, M.P. Modeling rational agents within a BDI architecture. In J. Allen, R. Fikes, and E. Sandewall, editors, Proceedings of the Third Internacional Conference on Principles of Knowledge Representation and Reasoning. Morgan Kaufmann (1991)
10. Bindiganavale R., Schuler W., Allbeck J., Badler N., Joshi A., Palmer P. Dynamically altering agent behaviors using natural language instructions. In Autonomous Agents Proceedings (2000)
11. Badler N., Bindiganavale R., Allbeck J., Schuler W., Zhao L, Palmer M. Parameterized Action Representation for Virtual Human Agents, in Embodied Conversational Agents, J. Cassell, J. Sullivan, S. Prevost, and E. Churchill, Eds. Cambridge, MA: MIT Press (2000) 256-284
12. Katoen J., Langerak R., Latella D. Modeling Systems by Probabilistic Process Algebra: An Event Structures Approach. Proc. on the IFIP TC6/WG6.1 Sixth International Conference on Formal Description Techniques, VI (1993) 253-268