A FRAMEWORK FOR NETWORKED EMOTIONAL CHARACTERS

DILZA SZWARCMAN
e-mail: dilza@inf.puc-rio.br

BRUNO FEIJÓ[†]
e-mail: bruno@inf.puc-rio.br

MÔNICA COSTA
e-mail: monica@inf.puc-rio.br

ICAD – Intelligent CAD Laboratory, Department of Computing, PUC-Rio
Rua Marquês de São Vicente, 225, 22453-900 Rio de Janeiro, RJ, Brazil

PUC-RioInf.MCC23/99 November, 1999

Abstract – This paper presents a framework for distributed emotional characters on a computer network and proposes innovative concepts for shared emotional state management. Firstly, behavior accuracy is defined in terms of three types of behavior (physical, procedural and emotional). Secondly, visual soundness – a concept inversely proportional to behavior accuracy – is associated with the autonomy of clones. Thirdly, the usual concept of dead reckoning is relaxed by exploring the idea of autonomy, which is the basis for networked emotional characters. The framework built on the top of Bamboo [7] can gracefully cope with reactive environments producing good levels of smooth animation, visual soundness, behavioral accuracy and collision avoidance.

Keywords – networked virtual environments, emotional characters, dead reckoning, behavior, autonomy.

Resumo – Este artigo apresenta um *framework* para desenvolvimento de ambientes virtuais distribuídos habitados por personagens com emoção e propõe novos conceitos para gerenciamento de estado compartilhado. Primeiramente, precisão de comportamento é definida em termos de três tipos de comportamento (físico, procedimental e emocional). Em seguida, solidez visual – um conceito inversamente proporcional à precisão de comportamento – é associada à autonomia de clones. Por último, o conceito usual de *dead reckoning* é relaxado, explorando a idéia de autonomia que é a base para personagens inteligentes distribuídos em rede. O *framework*, desenvolvido com a utilização da ferramenta Bamboo [7], suporta de forma elegante ambientes reativos produzindo bons níveis de suavidade de movimentos, solidez visual, precisão de comportamento e ausência de colisões.

Palavras-chave – ambientes virtuais distribuídos em rede, personagens com emoção, *dead reckoning*, comportamento, autonomia.

[†] Author for correspondence.

INTRODUCTION

Virtual environments distributed over computer networks began in the early 80's with the classified research on distributed military simulation by the US Department of Defense (SIMNET – simulator network) and networked games (notably *Flight* and *Dogfight* by SGI). However, despite all these years of activities, this research area still faces enormous challenges and lacks uniformity in its concepts and definitions. Notably, behavior, emotion, and autonomy of virtual humans are not well-defined concepts in the area of shared state management. Unfortunately, at the other side of the spectrum, most of the people working on emotional characters are not focused on networked virtual environments.

Emotional characters have been intensively investigated in the areas of behavioral animation[1], ecosystem simulation [2] and interactive drama in virtual worlds [3,4]. However, there is a very small number of works on the distribution of behavioral characters over computer networks. IMPROV [5] and JackMOO [6] are remarkable projects on distributed virtual actors, but they are not centered on the management of networked emotional states and cannot cope with reactive distributed environments.

This paper proposes a flexible and clear view of the concept of shared behavioral state management and presents a framework for networked emotional characters. A prototype is built on the top of BAMBOO [7], which is an open-architecture toolkit.

Before presenting the main ideas of this paper, some notes on terminology and general concepts are needed. Firstly the name of the whole area of research should be explained, because the literature often makes no distinction between "distributed virtual environments" and "networked virtual environments". Secondly, the techniques of managing shared state information are reviewed. After these preliminary explanations, the paper proposes the ideas of visual soundness, autonomy and recovering mechanisms as the basis for networked emotional characters. Finally, the framework is presented and conclusions are produced.

DISTRIBUTED OR NETWORKED VE ?

The term "distributed virtual environment" does not emphasize the nature of the problems to be tackled in this paper. Distributed virtual environment is a broader concept to characterize systems where an application is decomposed into different tasks that are run on different machines or as separate processes. A networked virtual environment is a more specific and adequate term to define a software system in which users interact with each other in real-time over a computer network. This is precisely the definition found in the new book by Sandeep Singhal and Michael Zyda [8], which is the first extensive reference organizing the many aspects in the field without being a documentation of standards. Sometimes the term "distributed interactive simulation environments" is used in the literature. This is motivated by the DIS network software architecture and its successor HLA (High Level Architecture) [9], which was recently been nominated as an IEEE standard and an OMG standard.

SHARED STATE MANAGEMENT

Shared state management is the area of networked virtual environment dedicated to the techniques of maintaining shared state information [8]. In a networked virtual environment, local objects are replicated in remote hosts and the users have the illusion of experiencing the same world. The consistency of this shared experience is hard to be achieved without degrading the virtual experience. When a state change is generated at a particular host, there is a necessary delay before all other hosts incorporate this update. One of the most important principles in networked virtual environments is that no technique can support both high *update rate* and absolute *state consistency*. This principle is a consequence of the characteristics of the computer networks, which are the latency, bandwidth and reliability. Network *latency* is the delay of transfer measured in units of time, network *bandwidth* is the rate of transfer (bits/second), and network *reliability* measures data packet losses. These network characteristics cannot be eliminated in any network technology, today or in the future. In order to guarantee total state consistency, hosts must always exchange acknowledgments and retransmit lost state updates before proceeding with new state updates.

In a networked virtual environment, at any given time, hosts will never produce identical views of the virtual world, unless centralized models [10] with absolute state consistency are implemented. Even with total state consistency, users will not see the same movement at the same time, because of the transmission time. Furthermore, in these centralized models, the source host may be forced to reduce its update rate to allow time for the consistency to be achieved. In any case of pursuing absolute consistency, the scene will not be realistic and the movement will not be smooth. Networked virtual environments must tolerate inconsistencies in the shared state updates.

There are two techniques for managing shared state updates that allow inconsistencies [8]: (1) *frequent state update notification*; (2) *dead reckoning*. In the first technique, hosts frequently broadcast (or multicast) update changes without exchanging acknowledgments. This technique has been mostly used by PC-based multiplayer games [11]. In dead reckoning techniques, the transmission of state updates is less frequent and the remote hosts predict the objects' behavior based on the last transmitted state.

The quality of a networked virtual environment is also determined by its *scalability*, which is its capacity of increasing the number of objects without degrading the system. Therefore techniques of managing shared state updates should be used in conjunction with resource management techniques for scalability and performance. Singhal and Zyda [8] present an excellent description of these later techniques, such as *packet compression*, *packet aggregation*, *area-of-interest filtering*, *multicasting protocols*, exploration of the human perceptual limitations (e.g. *level-of-detail perception* and *temporal perception*) and system architecture optimizations (e.g. *server clusters* and *peer-server systems*).

VISUAL SOUNDNESS AND BEHAVIORAL ACCURACY

In networked virtual environments the hosts should pass the *test of visual soundness* and every clone should exhibit *behavioral accuracy*.

The ***test of visual soundness*** is an expression proposed in this paper to indicate an important subjective test for networked virtual environments. A host passes the test of visual soundness if the user will be unable to distinguish between local and remote objects on the display. An environment that passes this test may be considered a networked virtual environment that is visually sound. This test fails when the movements of the clone are jerky or are composed of large jumps, which usually happen when the clone tries to follow the pilot's movement. Another reason of failure is the overlapping of objects, even when the animation is remarkably smooth.

A clone exhibits ***behavioral accuracy*** when it mirrors the following behaviors of its pilot:

- Physical behaviors (e.g. position, velocity, and acceleration);
- Procedural behavior (e.g. commitments, goals, and sequences of tasks);
- Emotional behaviors (e.g. fear, joy, and excitability).

It is evident that a host may exhibit a high level of behavioral accuracy but fails the test of visual soundness (e.g. clones and their pilots have quite similar positions and velocities, but clones' movements are jerky). The reverse is also possible; that is, a host may pass the test but exhibits low level of behavioral accuracy.

In fact, behavioral accuracy and visual soundness can be understood as being inversely proportional to each other. Behavioral accuracy depends on the system's shared state consistency. However, to increase state consistency, the update rates must be lowered, which in turn decreases visual soundness. Looking the other way around, visual soundness is related to local precision, which is achieved by giving *autonomy* to clones, that is, capacity to act by their own according to the local environment (view) presented by each host. This perspective based on the problem of autonomy is the starting point for the main ideas of this paper. A totally autonomous clone would act exactly like a local object but would, of course, have no behavioral accuracy. The challenge is to search for techniques that guarantee reasonable levels of behavioral accuracy while passing the test of visual soundness and providing good levels of scalability.

Dead reckoning is the basic technique to overcome the above-mentioned challenge. However, the usual concept of dead reckoning should be expanded. Firstly, dead reckoning protocols should be built on the basis of the clone autonomy. Secondly, these protocols should take into account the states of mind of entities and users. The consequences of this approach are many:

(1) The state of information may be any type of behavior (physical, procedural or emotional);
(2) There will be many different protocols adapted to each type of object and behavior;
(3) The protocol consists of two parts: a prediction mechanism and a recovery mechanism. The prediction mechanism estimates the values of the shared states and the recovery mechanism tries to restore values in the neighborhood of the pilots' state values.

It should be noted that total behavioral accuracy and perfect synchronism between pilots and clones are infeasible, even for simple scenes on a local high-speed network supporting high-end graphics workstations. We should assume that users would never see and experience the same virtual world in any circumstance - even in the situation of absolute consistency with very few users, such as two surgeons operating a remote patient. In this yet impossible dream of the networked virtual environment community, clones should take autonomous actions to guarantee local precision and the success of the operation.

CLONE AUTONOMY LEADING TO NETWORKED EMOTIONS

Dead reckoning has its origin in networked military simulations and multiplayer games, where users manipulate aircraft or ground vehicles. In these systems, prediction and recovery is a matter of trying to guess and converge to the pilot's trajectory. Second-order polynomial is the technique most commonly used to estimate these objects' future position based on current velocity, acceleration and position. It provides fairly good prediction since aircraft and vehicles do not usually present large changes in velocity and acceleration in short periods of time. Moreover, since the prediction algorithm is deterministic, the source host can calculate the error between pilot and clones' position, only sending updates when necessary. Convergence to the pilot's trajectory is usually done with curve fitting [8].

When it comes to articulated avatars, a couple of different approaches can be taken for dead reckoning. For applications requiring high state consistency, joint-level dead reckoning algorithms can be used to predict in-between postures [12]. This approach does not take into account the action the avatar is executing and dead reckoning computations are performed on joint angles information received at each update message. However, in many situations consistency at the level of articulated parts is not essential. Especially in collaborative work systems, where avatars interact with each other and together manipulate objects, visual soundness at each host becomes more important. Furthermore, articulated avatars have gained intelligence over the past years and efforts are being made to give them capacity to understand and accept orders like "*say hello to Mary*" [6]. In this context, dead reckoning at the level of physical actions has been considered [5, 6, 13]. That is, instead of sending position updates, the pilot sends to clones messages that indicate the low-level physical action it is executing: "*smile*", "*dance*" or "*wave*". In these systems, clones predict pilot's state by performing the same actions, regardless of the fact that each host may be presenting different objects movements. However, these script-based systems cannot cope with emotional and reactive behaviors.

This paper proposes another approach to avatar dead-reckoning that gives even more autonomy to clones, providing them with emotions and some power of decision. With no intelligence at all, clones are restricted to executing by their own only certain localized actions. For example, if the pilot tells them to "*walk to the door*", the environment will probably not remain still during all the time they take to get there. So, while they walk, the pilot must tell them how to react to environment changes – "*deviate right*" or "*get your head down*". However, remembering that pilot and clones always experience different views, some hosts may present poor visual soundness.

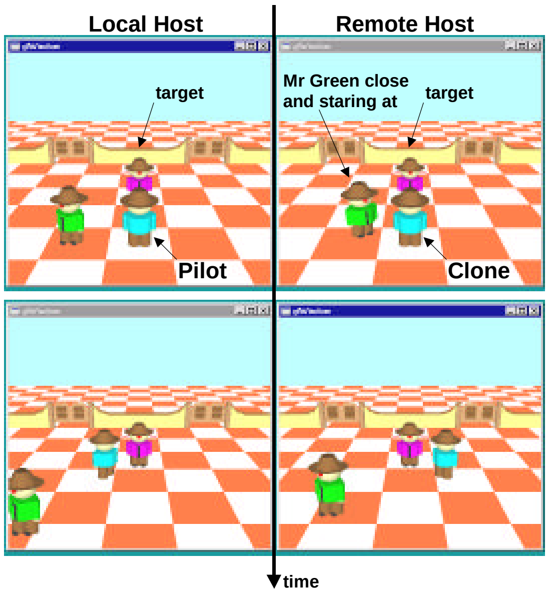


Fig. 1 Pilot and clone with different emotional states

On the other hand, if clones can make decisions based on what they *see*, *hear* and *feel* then they can get to the door naturally, deviating from an angry dog or smiling to a friend that passes by. Behavioral accuracy will be maintained at a higher level by making clones always consistent with pilot's overall commitments, goals, and emotional states. Clone autonomy is the only way to create an environment of networked emotions. Fig.1 shows the example of a character that does not like Mr. Green, one of the avatars in the room. In this example, the pilot and its clone experience different emotional state depending on the position and posture of Mr. Green and, consequently, they follow different paths towards the target. Although the local and remote hosts exhibit different animated scenes, this distributed virtual experience will be perfectly acceptable if the pilot has only the following characteristics: "*walk to the wall between the doors*" (procedural behavior) and "*I am afraid of Mr. Green*" (emotional behavior).

As in traditional dead reckoning approaches, the proposed management system for networked emotional characters is based on the tolerance to inconsistencies. Depending on the local view at each host, pilot and clones can decide differently. In some situations, different decisions can take clones' physical and emotional behavioral accuracy to unacceptable levels. In this matter, a recovery mechanism that restores state consistency is presented in the next section. It is worth noting that, for all types of dead reckoning algorithms, inconsistencies introduce undesirable consequences. For example, a dead reckoned aircraft might collide with another object during the convergence stage, when the predicted trajectory deviates from the real one. The literature lacks information about how these situations should be handled. This paper claims that the clones' autonomy is the most adequate approach to tackle these problems gracefully.

RECOVERING MECHANISM KEEPS CLONES UNDER CONTROL

Increasing the clones' autonomy makes them capable of reacting to the environment - avoiding collisions, expressing emotions and interacting with other avatars - according to what they experience locally. Each clone can act differently as long as it obeys the pilot's high level order. Put in another way, ***prediction becomes non-deterministic***. Taking again the "*walk to the door*" example, while the pilot decides to deviate right from a fearing entity, one or more clones may decide to deviate left or even not deviate at all, depending on the entity's state at each host. If all clones get to the door in approximately the same period of time, then the goal is achieved. However, clones that deviate left may encounter other obstacles that can make them get too far from the pilot and from the door. In this case, these clones should try to recover to the pilot's state.

Both, pilot and clone, have specific roles in the recovery mechanism. The pilot, after telling clones the task to be executed, should send them state update messages until the task is finished. The interval of time between updates is determined by environment properties that influence clones behavioral accuracy such as the type of application, the number of entities and the level of activity. In this way, the update rate will be dynamically adapted to environment demands. Since prediction is non-deterministic and the pilot has no way of knowing how far clones are, the update rate cannot be based on state error. Considering that updates have the purpose of helping clones out in extraordinary situations where they cannot find their way to execute a task, the average working update rate will tend to be low.

On the other hand, upon receiving an update message, each clone will verify if it needs to recover from inconsistencies. That will be the case if it is approximating neither the pilot state nor the goals. The clone will recover from inconsistencies by suspending the main task and executing a recovery action that will take it to the neighborhood of pilot's state in the most possible natural way. The recovery action is determined by a temporary redefinition of the priorities, which can eventually be more restrictive than the original top most priority (e.g. "converge to the actual pilot's path" takes precedence to the more generic task "leave the room"). When the recovery action is terminated, the main task is resumed.

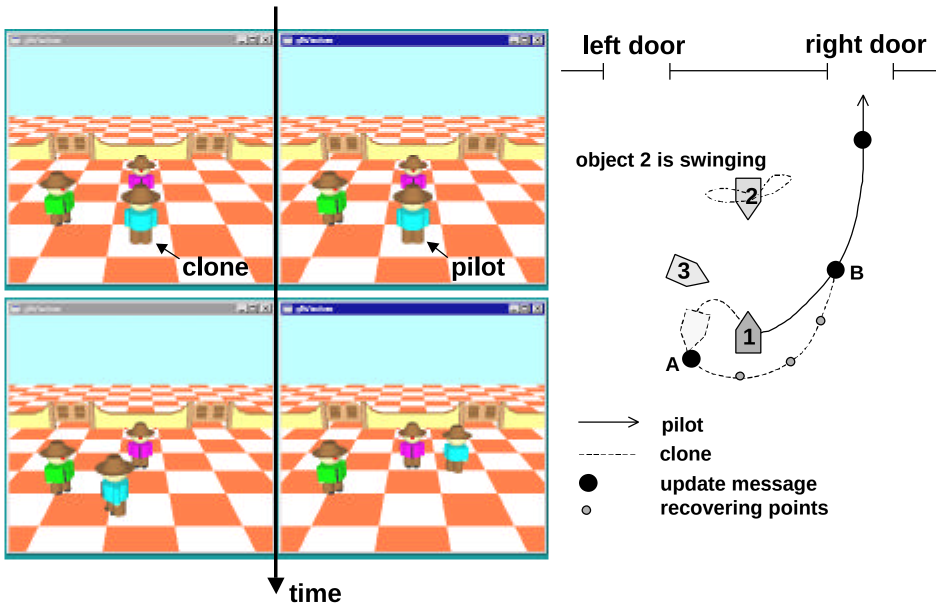


Fig 2 Recovering consistency

Fig. 2 illustrates the following case:

- ♦ The characteristics of the avatar 1 are "leave the room" and "keep away from people".
- ♦ Object 2 is swinging randomly around a small area, which causes different reactions from other objects (i.e. the hosts will never display the same position of object 2).
- ♦ The clone of the pilot 1 is initially going left (because the left door is a valid exit), when it notices object 3 and then starts getting far from the target.
- ♦ The update message from the pilot triggers the recovery action at point A.
- ♦ The clone 1 converges to the pilot's path.
- ♦ From point B forward anything can happen depending on the actual position of object 2 (e.g. clone 1 follows the pilot's path towards the right door, or clone 1 goes to the left door). In any case, the high-level behaviors will be consistent.

MODELING AN EMOTIONAL CHARACTER

Based on the architecture for emotional characters presented by [1] for behavioral animation, the mental model of Fig. 3 is proposed for pilots and clones in a networked environment. This model reflects a hybrid approach to avatars' intelligence that combines logical reasoning, proposed by traditional AI, with reactive behavior, the dynamic response to environment changes proposed by the agent theory. Its elements are drawn on general principles of cognition science that rule human mind models.

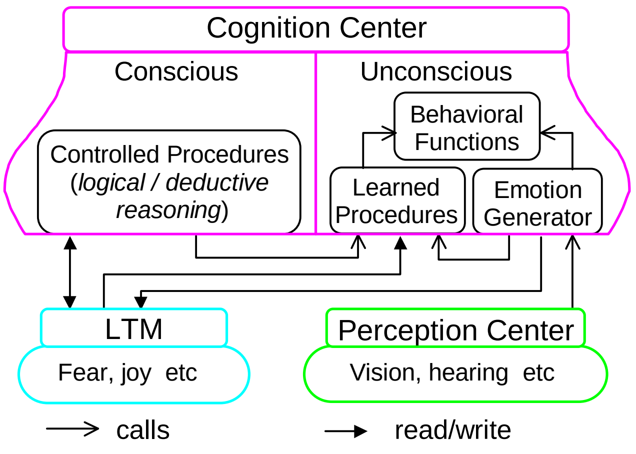


Fig. 3 A mental model for avatars

The Long Term Memory (LTM) stores facts and emotions, either pre-programmed or perceived by the character during its existence in the virtual environment. The data stored in the LTM is operated on by controlled and automatic procedures of the Cognition Center. Controlled procedures require conscious attention and are used for deductive and logical reasoning such as path-planning. Learned Procedures, Behavioral Functions and Emotion Generator are the automatic procedures that model the character's unconscious. Events or goals automatically trigger them.

Learned Procedures are reactive plans that continuously revise the LTM to adapt themselves to unexpected events and to the character's emotional state. The name Learned Procedures comes from the fact that these procedures represent learned skills with no need for conscious attention. Behavioral Functions are primitive forms of automatic procedures and represent reactive physical actions. The Emotion Generator operates on the LTM to generate emotional states.

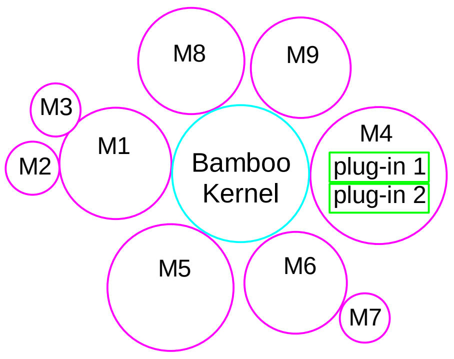
The Perception Center detects events in the virtual environment associated to vision, hearing and touch and passes the information to the Emotion Generator. The automatic procedures of the Cognition Center and the Perception Center together implement the reactive behavior of the avatar.

A FRAMEWORK ON TOP OF BAMBOO TOOLKIT

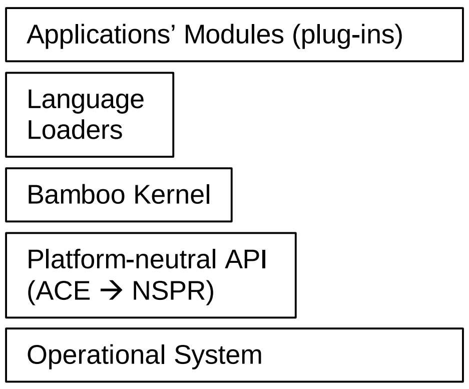
Bamboo is a language independent toolkit for developing dynamically extensible, real-time, networked applications [7]. Bamboo's design focuses on the ability of the system to configure itself dynamically, what allows applications to take on new functionality after execution. Although in its most recent versions Bamboo is presented as being application-neutral, it is particularly useful in the development of scalable virtual environments on the Internet because it facilitates the discovery of virtual components on the network at runtime. Bamboo's architecture is presented in Fig.4.

Provision for dynamic extensibility is accomplished by the implementation of the plug-in metaphor popularized by commercial packages such as Netscape Navigator. Bamboo offers a set of mechanisms that enable the coexistence of plug-ins in a multi-threaded environment. It also provides a particular combination of these mechanisms with a "main" routine, forming a specific executable called Bamboo's kernel, which constitutes the initial runtime environment for plug-ins to hook into. It is completely up to the plug-ins to give the application its functionality.

While plug-ins are the "units of extensibility", they are not the abstraction that Bamboo works on directly. Instead, Bamboo uses the "module", which can be thought of as a container of plug-ins. The data a plug-in inter-operates with can be physically coupled with it. Modules can then define geometry, behaviors, protocols and so forth. In practice, several modules will combine to create a specific application. Dynamically loaded modules can be retrieved from local disk or from the Web via HTTP.



(a) Bamboo's Abstract Runtime View


Applications' Modules (plug-ins)
Language Loaders
Bamboo Kernel
Platform-neutral API (ACE → NSPR)
Operational System

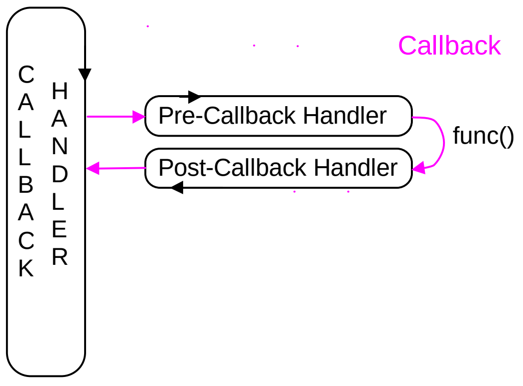
(b) Bamboo's Layering

Fig. 4 Bamboo's architecture

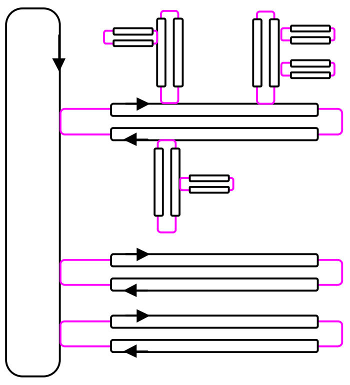
To provide portability, Bamboo's kernel sits on the top of a platform-neutral API for system calls, networking, threading, synchronization, and timing. Until the latest stable Bamboo's release [14], the Adaptive Communication Environment (ACE) library was used for that purpose. Bamboo's developers are now on the process of replacing ACE with NSPR, the Netscape Portable Runtime [15].

In the new version of Bamboo, language independence shall be provided through language-specific plug-in loaders that will: (1) load/unload its language's interpreter/virtual machine; (2) load/unload language-specific plug-ins for the kernel; (3) provide a language-specific API for kernel services.

Bamboo supports three kinds of plug-ins: internally driven, event driven, and externally driven. Internally driven plug-ins launch one or more threads of execution. Event driven plug-ins attach themselves to existing threads of execution. Finally, externally driven plug-ins do neither, but are utilized in some other way by the system, e.g. a math library. Event driven plug-ins are made possible by a Bamboo mechanism that allows *callbacks* to be attached to and removed from a *callback handler*. Callback handlers have the property of sequentially executing each of its callbacks when it itself is executed. Each callback is recursive in that it embeds two callback handlers, one just before and one just after the callback function is executed. In this way, the executable can be thought of as a tree of callbacks that can be dynamically extended in a very flexible and robust manner (Fig.5).



(a) Recursive Callbacks.



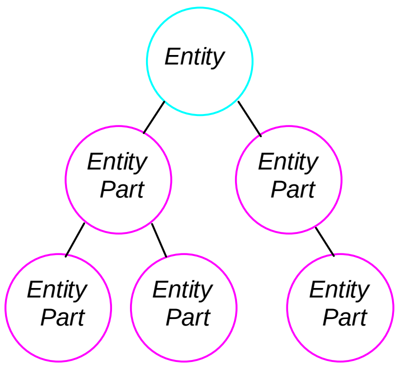
(b) Extending the Executable.

Fig. 5 Callback structure.

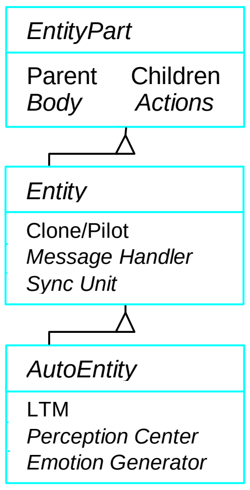
The proposed framework for distributed emotional characters is being developed as a C++ Bamboo module. It offers a set of classes that provides the basic functionality needed for shared behavioral state management. It also supports frequent state regeneration and the traditional form of dead reckoning for non-emotional entities.

Firstly, the framework considers that, at each host, the local virtual environment is composed of *Entities*, intelligent and non-intelligent ones. In other words, an *Entity* can be a static decoration object or a human-like avatar. Each shared component of the networked virtual world is then implemented by one *pilot Entity* and several *clone Entities*, one object at each participating host. The pilot, as the name suggests, is the master object whose state - considered the virtual component's true state - all clones continuously try to mirror. It is the pilot's responsibility to send messages to clones informing its state or the action it is executing so they can maintain behavioral accuracy at an acceptable level.

An *Entity* is actually the root node of a tree structure made up of *EntityParts* (Fig.6). The hierarchical structure reflects the physical dependence between parts: if one node moves, all its children move along. Every tree node may have a *Body* and a set of *Actions* it can perform. *Actions* are state machines that can be started, suspended, resumed or terminated.



(a) *Entity* Tree Structure.



(b) *Entity* Class Hierarchy.

Fig. 6 The Entity structure and hierarchy

The messages sent to an *Entity* are treated by its *Message Handler*. These messages can be remote or local ones (Fig. 7). Remote communication is only possible from a pilot to its clones or from one clone to its pilot. *Entities* pertaining to two different virtual components can communicate locally at each host. After receiving a local message from another component, a clone might need to inform its pilot about it.

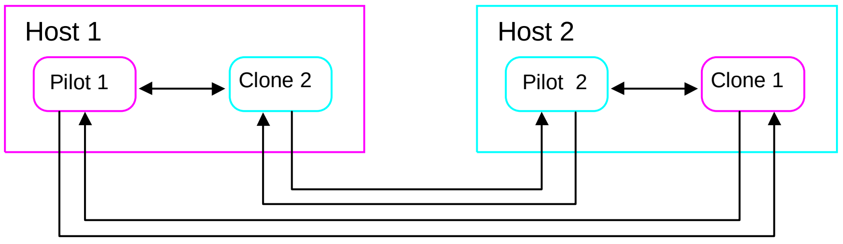


Fig.7 *Entities*' Communication.

The *Entity* object also includes a *Sync Unit* that is responsible for the shared state management. If the *Entity* is a pilot, the *Sync Unit* sends update messages to clones at the appropriate rate, depending on the type of virtual component and on environment properties. If it is a clone, the *Sync Unit* controls the activation of the recovery mechanism as updates messages arrive.

An intelligent *Entity*, called *AutoEntity*, yet includes a *Long Term Memory* (LTM), an *Emotion Generator* and a *Perception Center*, all parts of the mental model presented in Fig.3. *AutoEntities* implement emotional characters. In this case, the *Actions* that the *EntityParts* can perform correspond to behavioral functions. Learned procedures are *Actions* associated to the root node since they access the LTM and make no sense for individual parts. A set of sensor objects – *Vision* and others – constitute the *Perception Center*.

When the first *Entity* is created, a display thread, which continuously cycles the rendering engine, is launched. The rendering engine culls and draws all *Entities* on all active cameras and is implemented as a callback. Running *Actions* have their state machines cycled by attaching them as callbacks to the rendering engine's pre-callback handler, as shown in Fig.8. Sensors attach callbacks to the post-callback handler to verify environment changes that the current cycle of all *Actions* have created. Sensors' callbacks then activate the *Emotion Generators*, if necessary.

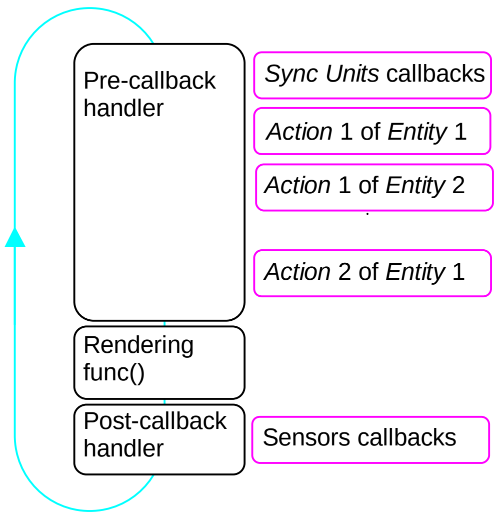


Fig. 8 Display thread

Sync Units, working in conjunction with the *Actions* class, also attach callbacks to the display thread to manage the virtual component's shared state. The mechanism is as follows:

1. The virtual component is ordered to perform an action that requires state consistency between pilot and clones (Action 1 in Fig.9). That could be, for example, to walk to the door or, in the case of a non-emotional avatar, to move according to a 3D mouse manipulation. The action is started by the pilot and is attached to the display thread.
2. Action 1 then executes enablePilotSync(), which attaches the *Sync Unit* callback. In the case of an emotional character, a message informing the action to be executed ("action1") is sent to clones. Action 1 is then attached to clones' display thread. Non-emotional clones do not execute Action1.
3. Clones' Action 1 executes enableCloneSync() to attach the *Sync Unit* callback to their threads. Non-emotional clones have their *Sync Unit* callback automatically activated by the first sync message received (step 4).
4. Pilot's *Sync Unit* callback continuously transmits state information in "sync_update", "sync_deadreck" or "sync_recover" messages, depending on the current sync mode. Sync mode is dynamically determined by environment conditions but it also depends on the virtual component's type.
5. Clones' *Sync Unit* callbacks verify incoming sync messages to determine the sync mode and coordinate *Actions* accordingly. If in update mode, clones are just updated. If in dead reckoning mode, Action 1 is suspended and the Dead Reckon Action, which corresponds to a position prediction algorithm, is attached. When convergence to the pilot's position is necessary, the Dead Reckon Action is suspended and the Convergence Action is started. Prediction is resumed when convergence terminates. In recover mode, which is only possible for emotional characters, Action 1 is suspended and Recover Action attached whenever clones need to recover to pilot's state.

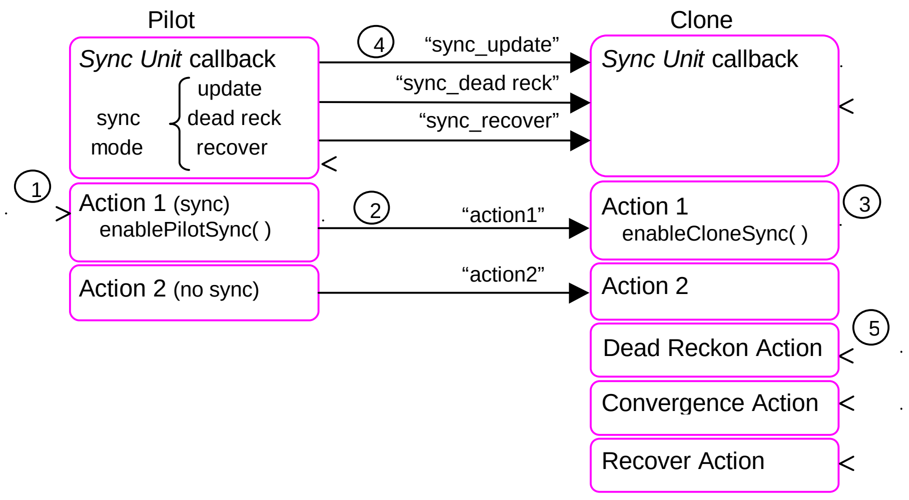


Fig. 9 Shared state management.

When Action 1 terminates it detaches the *Sync Unit* callback before it removes itself from the display thread.

An action that does not require state consistency (Action 2 in Fig. 9) could be the movement of a clock's pendulum, the rotation of a propeller or a human virtual eating a sandwich. In that case, the pilot sends to clones only a message informing the action to be performed.

The above mechanism makes shared state management highly flexible in the sense that it can adapt itself to the virtual component's type, emotional or non-emotional, to the actions type, with or without state consistency, and to dynamic environment conditions such as the activity level.

The framework also provides the Network Interface Unit (*NIU*) to take care of remote communication at the virtual environment level. Incoming messages related to the virtual environment's composition and configuration, such as creation and deletion of clones, are handled by the *NIU* itself. Component level messages are passed on to the appropriate local *Entity*. The *NIU* is able to deliver a message to the right clone because it maintains a map associating pilots' host addresses to corresponding local clone *Entities* and uses the message's *from* address as an index to the map.

CONCLUSIONS AND FUTURE WORK

Behavior, emotion, and autonomy of virtual humans are not well-defined concepts in the area of networked virtual environments. On the other hand, most of the people working on emotional characters are not focused on shared state management over a computer network. This paper explores a common view amongst these areas of research and proposes innovative concepts for shared emotional state management. Firstly, behavior accuracy is defined in terms of three types of behavior (physical, procedural and emotional). Secondly, visual soundness – a concept inversely proportional to behavior accuracy – is associated with the autonomy of clones. Although the literature have already mentioned that dead reckoning should handle any type of shared state and state prediction may be object-specific [8], these ideas are not clearly explained. For instance, the authors of Improv [5] do not present this system in terms of shared state management, although its scripted events have been recognized as a form of object-specific state prediction elsewhere [8]. JackMOO [6] is also another impressive form of scripted events, but like Improv it cannot cope with reactive environments and dynamic emotional characters. As far as collision is concerned, real-time distributed collision agreement for dead reckoning algorithms are not clearly mentioned in the literature [8]; and the existent systems probably fail the test of visual soundness. Therefore, most of the interesting dead reckoning extensions produces smooth animation but cannot cope with undesirable collisions that happen during the convergence stage, such as the position history-based protocol [16]. One possible solution for many of the above-mentioned drawbacks is to relax the concept of dead reckoning by exploring the idea of autonomy, which is the basis for distributed emotional characters on a computer network. Therefore, this paper proposes a framework for networked emotional characters that can also cope with reactive environments presenting good levels of visual soundness, behavioral accuracy and collision avoidance.

Several examples were run in a Windows NT environment producing good results, although performance was not formally investigated. Some of these examples are presented in Figs. 1 and 2 above.

There are several topics for future work. The framework should be expanded and improved to handle more complex animation and characters. The mental model for avatars is surely a subject for further investigation. Recovering criteria should also be studied in more detail – on the pilot's side, this concerns the set of rules used to dynamically adapt the update rate and, on the clones' side, the set of rules used to trigger the recovering action. Also resource management techniques for scalability and performance should be investigated, especially *area-of-interest filtering* that is central to networked emotional characters. Real-time distributed collision agreement is another important topic for further research.

Acknowledgments – The authors would like to thank the CNPq for the financial support, which includes scholarships (doctoral, post-doctoral and individual research) and equipment financing through the Pronex program.

REFERENCES

[1] M.Costa and B.Feijó, Agents with emotions in behavioral animation. *Comput. & Graphics*, **2**, No. 3, 377-384 (1996).
[2] X. Tu and D. Terzopoulos, Artificial fishes: physics, locomotion, perception, behavior. In *Computer Graphics Proceedings, Annual Conference Series, ACM SIGGRAPH*, 43-50 (1994).
[3] J. Bates, A.B. Loyall and W.S. Reilly, Integrating reactivity, goals, and emotion in a broad agent. *Technical Report CMU-CS-92-144*, School of Computer Science, Carnegie-Mellon University, Pittsburgh, PA (1992).
[4] H. Maldonado, A. Picard, P. Doyle and B. Hayes-Roth, Tigrito: a multi-mode interactive improvisational agent. *Stanford Knowledge Systems Laboratory Report KSL-97-08*, Stanford University (1997).
[5] K. Perlin and T. Goldberg, Improv: a system for scripting interactive actors in virtual worlds. In *Computer Graphics Proceedings, Annual Conference Series, ACM SIGGRAPH*, 205-216 (1996).
[6] J. Shi, T.J. Smith, J. Granieri and N.I. Badler, Smart avatars in JackMOO. In *Proceedings of the 1999 Virtual Reality Conference (VR'99)*, IEEE, Houston, Texas, USA, 156-163 (1999).
[7] K. Watsen and M. Zyda, Bamboo – a portable system for dynamically extensible, networked, real-time, virtual environments. In *Proceedings of the 1998 Virtual Reality Annual International Symposium (VRAIS'98)*, IEEE, Atlanta, GA, 252-259 (1998).
[8] Singhal, S. and Zyda, M., Networked Virtual Environments – Design and Implementation. ACM Press, 1999.
[9] J. Dahmann, R. Weatherly and F. Kuhl, *Creating Computer Simulation Systems: An Introduction to the High Level Architecture*, Prentice-Hall, Upper Saddle River, NJ (1999).
[10] Anupam, V. and Bajaj, C., Distributed and collaborative visualization. IEEE Multimedia, 1(2), p. 39-49, Summer 1994.
[11] Id Software, Doom, Dec. 1993.
[12] T.K. Capin, I.S. Pandzic, D. Thalmann, N. Magnenat Thalmann, A Dead-Reckoning Algorithm for Virtual Human Figures. In *Proceedings of the 1997 Virtual Reality Annual International Symposium (VRAIS'97)*, IEEE, Albuquerque, USA, 161-169 (1997).
[13] Living Worlds Web site , http://www.vrml.org/WorkingGroups/living-worlds/draft_2 , (October 10, 1999).
[14] Old Bamboo Web site, http://watsen.net/Bamboo-old, (March 3, 1999).
[15] Bamboo Web sites, http://www.npsnet.nps.navy.mil/Bamboo/, http://watsen.net/Bamboo/, (October 10, 1999).
[16] S.K. Singhal and D.R. Cheriton, Using a position history-based protocol for distributed object visualization, Chapter 10 of Designing Real-Time Graphics for Entertainment [Course Notes for SIGGRAPH '94 Course #14] (July 1994). Also available as Technical Report STAN-CS-TR-94-1505, Department of Computer Science, Stanford University (February 1994).