A Framework for Networked Reactive Characters

DILZA SZWARCMAN†
BRUNO FEIJÓ
MÔNICA COSTA
ICAD – Intelligent CAD Laboratory, Department of Informatics, PUC-Rio,
Rua Marquês de São Vicente, 225, 22453-900 Rio de Janeiro, RJ, Brazil
e-mail: dilza@inf.puc-rio.br

Abstract – This paper presents a framework for distributed reactive characters on a computer network and proposes innovative concepts for shared state management. Firstly, behavioral accuracy is defined in terms of three types of state (physical, procedural and emotional). Secondly, visual soundness – a concept inversely proportional to behavioral 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 reactive characters. The framework, built on top of an open-architecture toolkit called Bamboo, can gracefully cope with reactive environments producing good levels of visual soundness, which results in smooth animations, and behavioral accuracy.

1 Introduction

Virtual environments distributed over computer networks began in the early 80's [1]. However, despite all these years of activities, this research area still faces enormous challenges and lacks uniformity in its concepts and definitions, specially considering networked virtual environments populated by reactive characters. Notably, autonomy of virtual humans is a subject not fully investigated in the context of shared state management. Unfortunately, at the other side of the spectrum, most of the people working on reactive characters [2, 3, 4, 5, 6, 7, 8] are not focused on networked virtual environments.

JackMOO [9], VLNET [10] and Improv [11] are remarkable projects on distributed virtual actors, however, the first two adopt centralized models and thus have limited scalability, while the latter does not concentrate on shared state management.

This paper relies on clone autonomy as an adequate approach to networked virtual environments inhabited by reactive characters. It proposes a flexible and clear view of the concept of shared state management and presents a framework for networked reactive characters. A prototype is built on top of Bamboo [12, 13], which is an open-architecture toolkit particularly useful in the development of scalable virtual environments.

Before presenting the main ideas of this paper, the techniques for managing shared state information are described in section 2. After this preliminary review, section 3 presents visual soundness and behavior accuracy as desirable properties of a networked virtual environment. It also proposes dead reckoning on the basis of clone autonomy as an adequate shared state management technique to obtain good levels of the above properties. Sections 4 and 5 explain in detail the proposed technique, starting out with a summary of other types of dead reckoning algorithms available. In section 6, the paper presents the model of reactive characters used. Although the model constitutes a fundamental part of the framework's development, it is not the main focus of this paper. The resulting framework's class structure is presented in section 7. Finally, experimental results are discussed in section 8 and conclusions are produced in section 9.

2 Shared State Management

Shared state management is the area of networked virtual environment dedicated to the techniques of maintaining shared state information [1]. 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. In fact, one of the most important principles in networked virtual environments says that no technique can support both high *update rate* and absolute *state consistency* [1]. To guarantee total consistency, the state of the virtual

† Author for correspondence.

0-7695-0878-2/00 $10.00 © 2000 IEEE

environment can only be changed after the last update has been surely disseminated to all hosts. For that matter, shared state information must be stored in a centralized repository – or some corresponding model [14] – with hosts exchanging acknowledgements and retransmitting lost state updates before proceeding with new ones. Update rates, therefore, become limited which in turn produces non-smooth movements and non-realistic scenes. Highly dynamic virtual environments, such as the ones populated by reactive characters, must tolerate inconsistencies in the shared state.

There are two techniques for managing shared state updates that allow inconsistencies [1]: (1) *frequent state update notification* and (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 multi-player games [15]. In dead reckoning techniques (see section 5), the transmission of state updates is less frequent and the remote hosts predict the objects' behavior based on the last transmitted state. Both techniques allow the use of decentralized replicated databases where every virtual component has a *pilot*, the main representation that resides in its controlling host, and several *clones*, the replicas that reside in other hosts.

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 [1] present an excellent description of some of these techniques, such as *packet compression*, *packet aggregation*, *area-of-interest filtering* and *multicasting protocols*.

3 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*. These two concepts are proposed in this paper and are presented below.

A host passes the ***test of visual soundness*** if all virtual components move smoothly and interact in a precise way (local precision). One example of local precision failure is the overlapping of objects. Another example is when a character fails to put his hand in the right place while trying to shake hands with another one. An environment that passes the test of visual soundness may be considered a networked virtual environment that is visually sound.

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

- physical (e.g. position, velocity, and acceleration),
- procedural (e.g. commitments, goals, and sequences of tasks),
- emotional (e.g. fear, joy, and excitability).

It is evident that a host may exhibit a high level of behavioral accuracy but fail 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 exhibit 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, increasing state consistency limits the update rate, 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 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 clone autonomy. Secondly, these protocols should take into account the states of mind of entities. The consequences of this approach are many:

(1) The state information may be physical, procedural or emotional.
(2) There will be many different protocols adapted to each type of object and behavior.
(3) The protocols consist 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.

4 Networked Reactive Clones

Dead reckoning has its origin in networked military simulations and multi-player 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, the source host can also execute the prediction algorithm and 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 [1].

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 [16]. This approach does not take into account the action the avatar is executing and dead reckoning computations are performed on joint angle 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, local precision 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*" [9]. In this context, dead reckoning at the level of physical actions has been considered [9, 10, 11, 17]. 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 object movements. However, in these script-based systems clones reactivity is very limited. For instance, clones cannot deviate from other objects if their pilots do not order them to. They can decide, though, the best place to put their foot when stepping forward.

This paper proposes another approach to avatar dead reckoning that gives even more autonomy to clones, providing them with 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. 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.

Figure 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 states 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).

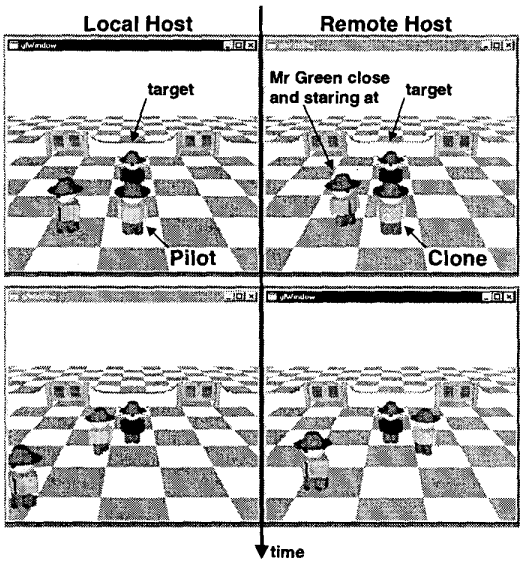

Figure 1 Pilot and clones with different emotional states.

As in traditional dead reckoning approaches, the proposed management system for networked reactive 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 when the predicted trajectory deviates from the real one. The literature lacks information about how these situations should be handled. This paper claims that, for reactive

characters, clones' autonomy is the most adequate approach to tackle these problems gracefully.

From the above reasoning, the proposed framework was designed to support dead reckoning of high level tasks for reactive characters and traditional dead reckoning for non-reactive avatars. Physical action dead reckoning is made available as a special case of high-level task dead reckoning, namely as a task that does not require recovery. The framework's shared state management mechanism is highly flexible in the sense that it can adapt itself to the virtual component's type and to the task type, with or without recovery.

5 Recovering Mechanism Keeps Clones under Control

Increasing clones' autonomy makes them capable of 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. The prediction mechanism is such that the estimated states are not the same at all hosts and are unknown to the pilot. 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. A recovery mechanism is proposed for this purpose.

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 may be 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 predicted states are not known to the pilot, 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 natural way possible. Recovery is determined by a temporary redefinition of 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.

Figure 2 illustrates the following case:

- The characteristics of 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).

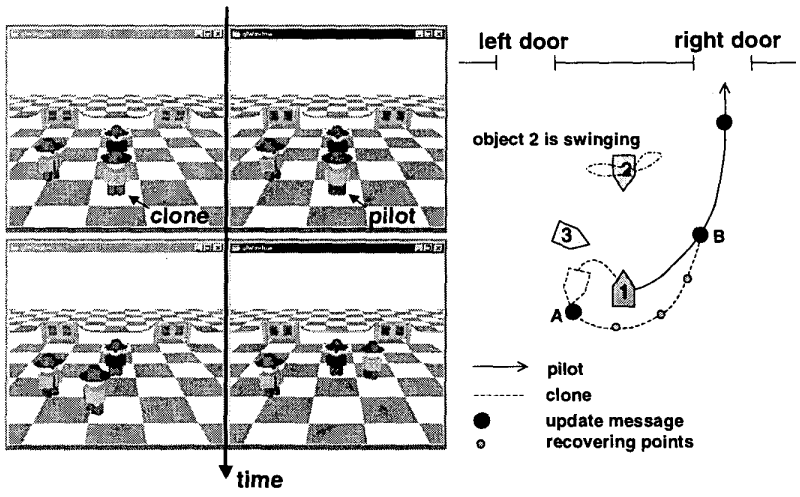

Figure 2 Recovering consistency.

- The clone of 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.
- 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.

6 Modeling a Reactive Character

Based on the architecture for reactive characters presented by [5] for behavioral animation, the mental model of Figure 3 is proposed for pilots and clones in a networked virtual 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.

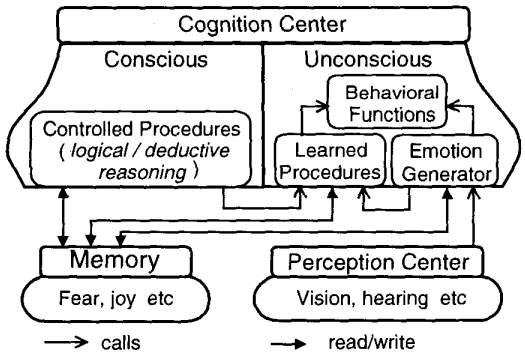

Figure 3 A mental model for avatars.

The Memory stores facts and emotions, either pre-programmed or perceived by the character during its existence in the virtual environment. The data stored in the Memory 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 Memory 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 Memory 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.

7 A Framework on Top of Bamboo Toolkit

Bamboo is a toolkit for developing dynamically extensible, real-time, networked applications [12, 13]. Bamboo's design focuses on the ability of the system to configure itself dynamically, what allows applications to take on new functionality after execution. Bamboo 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.

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 (Figure 4). It is completely up to the plug-ins to give the application its functionality.

Bamboo works with "modules", which can be thought of as containers 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.

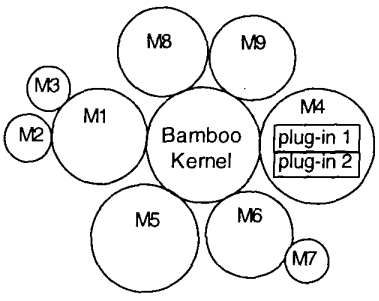

Figure 4 Bamboo's abstract runtime view.

The proposed framework for distributed reactive characters is being developed as a C++ Bamboo module. It offers a set of classes that provides the basic functionality needed for shared state management as described in sections 4 and 5. It also supports frequent state regeneration.

Firstly, the framework considers that, at each host, the local virtual environment is composed of *Entities*, reactive and non-reactive 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 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* (Figure 5). 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.

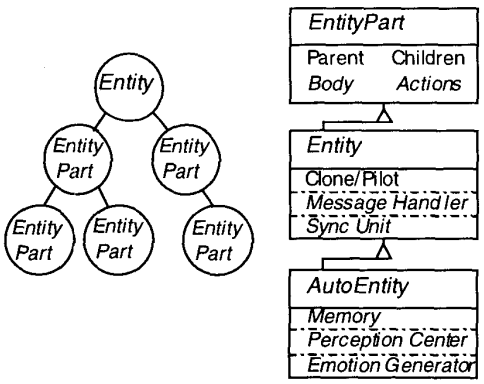

Figure 5 The Entity structure and hierarchy.

The messages sent to an *Entity* are treated by its *Message Handler*. These messages can be remote or local ones. 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.

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 whether state consistency is maintained with frequent state notification or with traditional dead reckoning. If it is a clone, the *Sync Unit* controls the activation of dead reckoning prediction and convergence algorithms as update messages arrive.

A reactive *Entity*, called *AutoEntity*, yet includes a *Memory*, an *Emotion Generator* and a *Perception Center*, all parts of the mental model presented in Figure 3. 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 *Memory* and make no sense for individual parts. A set of sensor objects – *Vision* and others – constitute the *Perception Center*. The *AutoEntity* class has its own *Sync Unit* that overrides the corresponding unit in the *Entity* class. If the *AutoEntity* is a pilot, the *Sync Unit* sends update messages to clones at the appropriate rate, depending on whether state consistency is maintained with frequent state notification or with dead reckoning of high-level tasks. In the case of dead reckoning, the automatic adaptation of the update rate to environment demands is not yet part of the framework though it can be programmed by the application developer. If it is a clone, the *Sync Unit* controls the activation of the recovering mechanism – suspension of the main task and execution of the recover action – as update messages arrive.

When the framework module is loaded, 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 Figure 6. Sensors attach callbacks to the post-callback handler to verify environment changes that the current cycle of all

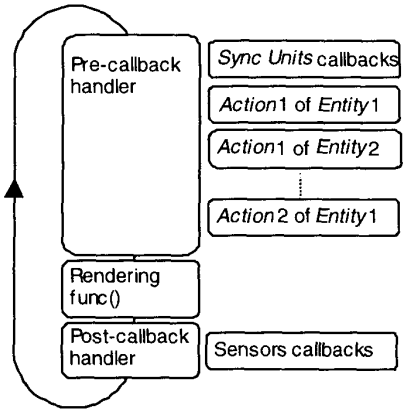

Figure 6 Display thread.

Actions have created. Sensors' callbacks then activate the *Emotion Generators*, if necessary. *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 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*.

8 Experimental Results

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 Figures 1 and 2 above.

Figure 7 shows another example where Mr. Green is ordered to move to a position four blocks straight ahead. A flying disc that is managed by traditional dead reckoning comes in the way. Mr. Green's pilot and clone take on different paths depending on the position of the disc at each host. However, collision is avoided and Mr. Green gets to the desired position at both hosts.

In addition to the good levels of visual soundness produced, an important result is the reduced number of transmitted networked messages. As already mentioned, to "walk to the door" a clone with limited autonomy would require one message for each step of the pilot's way. Considering a rather low velocity, this would mean 1 message/sec. In the prototype environment shown, one recover message every 7 seconds produced good behavioral accuracy for a level of activity of four moving entities. This primary result gives a fairly good idea of the benefits of the proposed management mechanism.

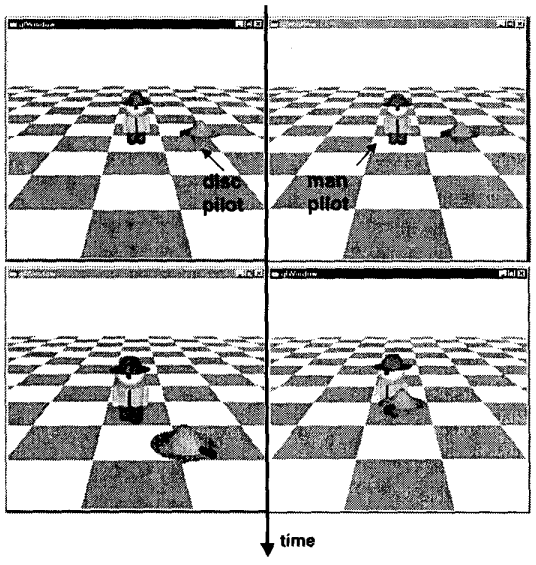

Figura 7 Reactive and non- reactive entities

9 Conclusions and Future Work

Behavior and autonomy of virtual humans are not well explored concepts in the area of networked virtual environments. On the other hand, most of the people working on reactive 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 state management. Firstly, behavioral accuracy is defined in terms of three types of behavior (physical, procedural and emotional). Secondly, visual soundness – a concept inversely proportional to behavioral 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 [1], these ideas are not clearly explained. For instance, the authors of Improv [11] 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 [1]. JackMOO [9] is another impressive form of scripted events system, but like in Improv clones have very limited reactivity. As far as collision is concerned, real-time distributed collision agreement for dead reckoning algorithms are not clearly mentioned in the literature [1]; the existent systems probably fail the test of visual soundness. Most of the interesting dead reckoning extensions, such as the position history-based protocol [18], produce smooth animation but cannot cope with undesirable collisions that happen when the predicted trajectory differs from the real one. For reactive characters, one possible solution for many of the above-mentioned drawbacks is to relax the concept of dead reckoning by exploring the idea of autonomy. Therefore, this paper proposes a framework for networked reactive characters that supports dead reckoning of high level tasks producing good levels of visual soundness and behavioral accuracy.

The proposed shared state management mechanism also works in favor of scalability in that it substantially reduces the number of transmitted network messages. Moreover, as virtual characters become more intelligent the average recovery messages rate becomes lower. Clones tend to need pilot's help less frequently. Naturally, reduced message rate implies greater local computation.

Some of the drawbacks of physical action dead reckoning are not solved by dead reckoning of high-level tasks. One of them is related to the interaction between reactive characters and moving objects. If, for example, while "Mr. Green" is dancing, a ball is thrown, it might hit "him" at one host and not hit "him" at another host. Either the ball's trajectory is allowed to be different at each host, which poses a problem since the ball will not be able to "walk" to the right position, or the ball is forced to always follow its pilot's path which will produce unrealistic scenes. Another drawback is related to the initialization of actors when a new participant joins the environment. As existing actors may be executing an action, initialization of clones at the new user's host becomes more elaborate then a simple state update. For environments divided in areas of interest associated to multicast groups, initialization is an important issue because clones may have to be initialized as the pilot crosses region boundaries.

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,* which is central to networked reactive 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 (scholarships and equipment financing through the Pronex program) and the SIBGRAPI 2000 reviewers for the valuable contributions.

References

[1] S. Singhal and M. Zyda, *Networked Virtual Environments – Design and Implementation*, ACM Press (1999).

[2] C.W. Reynolds, "Flocks, herds, and schools: a distributed behavioral model". In *Proc. of SIGGRAPH'87*, volume 21, 25-34 (July 1987).

[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] X. Tu and D. Terzopoulos, "Artificial fishes: physics, locomotion, perception, behavior". In *Computer Graphics Proceedings,* Annual Conference Series, ACM SIGGRAPH, 43-50 (1994).

[5] M. Costa and B. Feijó, "Agents with emotions in behavioral animation", *Comput. & Graphics* 2, No. 3, 377-384 (1996).

[6] 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).

[7] N. Badler, R. Bindiganavale, J. Bourne, M. Palmer, J. Shi and W. Schuler, "A parameterized action representation for virtual human agents". In *Workshop on Embodied Conversational Characters*, Lake Tahoe, CA (1998).

[8] P. Béicheiraz, D. Thalmann, "A behavioral animation system for autonomous actors personified by emotions". In *Proc. of First Workshop on Embodied Conversational Characters* (October 1998).

[9] J. Shi, T.J. Smith, J. Granieri and N.I. Badler, "Smart avatars in JackMOO". In *Proc. of the 1999 Virtual Reality Conference (VR'99)*, IEEE, Texas, USA, 156-163 (1999).

[10] T.K. Capin, I.S. Pandzic, N.M. Thalmann and D. Thalmann, "Realistic avatars and autonomous virtual humans in VLNET networked virtual environments". In *Virtual Worlds in the Internet* (R. Earnshaw and J. Vince, eds.), Chapter 8, IEEE Computer Society Press, 157-174 (1998).

[11] 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).

[12] K. Watsen and M. Zyda, "Bamboo – a portable system for dynamically extensible, networked, real-time, virtual environments". In *Proc. of the 1998 Virtual Reality Annual International Symposium (VRAIS'98)*, IEEE, Atlanta, GA, 252-259 (1998).

[13] Bamboo Web site (as in April 5, 2000), http://npsnet.org/~watsen/Bamboo/index.html.

[14] V. Anupam and C. Bajaj, "Distributed and collaborative visualization". *IEEE Multimedia* 1(2), 39-49 (Summer 1994).

[15] Id Software, Doom (Dec. 1993).

[16] T.K. Capin, I.S. Pandzic, D. Thalmann, N. Magnenat Thalmann, "A dead-reckoning algorithm for virtual human figures". In *Proc. of the 1997 Virtual Reality Annual International Symposium (VRAIS'97)*, IEEE, Albuquerque, USA, 161-169 (1997).

[17] Living Worlds Web site (as in October 10, 1999), http://www.vrml.org/WorkingGroups/living-worlds/draft_2

[18] 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).