A Migration Mechanism to Manage Network Troubles while Interacting within Collaborative Virtual Environments

Thierry Duval*
IRISA
Campus universitaire de Beaulieu
F-35042 Rennes Cedex — France

Chadi el Zammar†
VDL2
700, rue Wellington — bureau 1200 — H3C 1T4
Montréal (Québec) — Canada

Abstract

Real time collaborative interactions within CVEs rely on the low latency offered by high speed networks. When low level network troubles occur during a collaborative session, participants of networked collaborative virtual environments can suffer from misunderstanding weird behavior of some objects of the virtual universe. We want to make such a virtual world easier to understand by using some graphic visualizations in such a way that the users become aware of these problems. It is a kind of augmentation of the Virtual Environment thanks to dedicated 3D metaphors associated to some objects of the virtual environment. We also want to allow users to work while these troubles occur, maybe with only restricted interaction possibilities, by making some interactive objects migrate onto the same site than the user who wants to interact with them. This is why we present then how two independent mechanisms may be coupled together for a better management and awareness of network troubles while interacting within a networked collaborative virtual environment: an awareness system that visualizes, through special metaphors, the existence of a network trouble as strong delay or disconnection; and a virtual object migration system that allows the migration of an object from one site to another to ensure a non interrupted manipulation in case of network troubles.

CR Categories: H.5.1 [INFORMATION INTERFACES AND PRESENTATION]: Multimedia Information Systems—Artificial, augmented, and virtual realities; H.5.3 [INFORMATION INTERFACES AND PRESENTATION]: Group and Organization Interfaces—Computer-supported cooperative work I.3.7 [COMPUTER GRAPHICS]: Three-Dimensional Graphics and Realism—Virtual reality I.3.2 [COMPUTER GRAPHICS]: Graphics Systems—Distributed/network graphics

Keywords: Collaborative Virtual Environments, Computer-Supported Cooperative Work, Virtual Reality, Augmented Virtuality, Awareness of Network Problems, Object Migration.

1 Introduction

Manipulation of objects is the most fundamental tasks of 3D interaction in Virtual Reality (VR) [Bowman. D. et al. 2004], it means that a user can manipulate and control objects. Within a collaborative virtual environment, an interactive object may either be controlled by one user in an exclusive way or by several users in a collaborative way. In both cases the evolutions of the shared environment are more difficult to understand than in a single-user environment because these evolutions can be caused by the other users. It is even more difficult when the evolution is caused by collaborative interactions. These collaborative interactions can occur in real time thanks to the low latency offered by high speed networks. This is the reason why many researchers focus on making users aware of other users' interactions with the objects of the shared virtual worlds [Gutwin, C. and Greenberg, S. 1998] [Fraser, M. et al. 1999] [Fraser, M. et al. 2000]. These studies consider that data transmission over the network is always reliable, so they never matter about the problems that could arise if there were network troubles. However, when network troubles happen such as the increase of the network delay or the disconnection of some sites, some problems of consistency can arise between the different instances of the shared universe. We need visual metaphors to make the users aware of these inconsistencies and rescue solutions to allow them to go on working anyway. They must be aware that there is no possibility to interact with a set of objects, or that actions of some other users are not updated any more, so that they can go on interacting with objects of the world that are not disconnected from their site.

We will focus on networked virtual environments that connect a few number of participants who collaborate closely to achieve a specific task, for example to assemble some parts of a car mockup [Duval, T. and Le Tenier, C. 2004]. With such constraints, a network perturbation or disconnection can cause damages to the remote interactions that users are performing. The solutions usually proposed in this domain do not resolve this problem in a global way, despite they try to reduce the effect of network troubles by reducing the amount of circulating updates and messages on the network. These solutions are not sufficient as long as networks are the base structure of distributed virtual environments. This is why we look at this problem from a different point of view: we admit that these network troubles can arise so we have to make all the users of a collaborative virtual environment aware of them. This way, users may understand and deal with some temporarily network troubles, especially while they are interacting with virtual objects. Furthermore, thanks to virtual object migration, we want to prevent a user from losing the control of a remotely calculated object in case of network problem.

2 Related work

2.1 Network delays and side effects

The network affects directly the performance of distributed virtual environments systems. For example when interacting remotely, a user can take the control of a virtual object and manipulate it. Low latency offers real time interaction by minimizing the delay between user's actions and distant object's responses. A data transmission problem on the network will directly affect the remote interaction. Most frequently types of problems envisaged are: delay when transmitting data over a network [Vaghi, I. et al. 1999][Fraser, M. et al. 2000]; short disconnections from time to time due to dynamic rerouting systems; or temporarily breakdown of the network [Macedonia, M. et al. 1994]. Consequences of the increase of the

*e-mail: Thierry.Duval@irisa.fr
†e-mail: Chadi.Zammar@gmail.com

disconnection time of a site can be catastrophic because the users can not understand any longer the global behavior of the shared virtual universe [Gutwin, C. et al. 2004], and several studies has been done to try to manage collaborative interactions even with some network delay as detailed in [Gutwin, C. 2001] and [Park, K. and Kenyon, R. 1999]. So, from the interactivity point of view, when a network disconnection occurs, the user's interactions that exist on the disconnected site are not seen any more by the other sites. This same user is not able any more to see the interactions of the other users and it also produces collaboration breakdowns.

When we can predict an object evolution, we can tone down the communication problem by using a prediction system that achieves local computing to estimate the future status of an object. NPSNET implements this method using the "Dead Reckoning" algorithm [Macedonia, M. et al. 1994]. But when we are facing totally unpredictable actions like users' interactions, such prediction systems are useless. Some other systems like SPIN [Dumas, C. et al. 1999] duplicate all universes objects and perform parallel calculations locally on each node. In this case network delay is not meaningful from the animation point of view, but this architecture does not resolve the main problem when objects are interactive, because users' interactions can not be "duplicated" any longer during a network breakdown. OpenMASK[1] [Margery, D. et al. 2002] implements a predictive architecture based on the concept of referentials and mirrors similar to the distribution concept in NPSNET. A referential is an independent entity that can evolve through internal state calculus. A mirror is a light copy of a referential, its evolution relies completely on updates received from its associated referential. This referential/mirror paradigm is the base of our work.

2.2 Providing awareness of network troubles

[Vaghi, I. et al. 1999] experimented a collaborative two players ball game where one player was subjected to an increasing amount of delay. They observed that as the network delay increases, the users (being aware) modify their strategies in an attempt to cope with the situation. [Fraser, M. et al. 2000] made a step forward by giving visibility to what they have called "delay induced phenomena". They have implemented a system that estimates the maximum difference between the "objective" position of a user avatar, and where another might perceive it to be, accordingly to network delay times between the users and speed of motion. For example, in case of network delay, an avatar is shown surrounded by a sphere that represents all uncertain possible positions, and this sphere evolves accordingly to the network delays. [Gutwin, C. et al. 2004] proposed visual metaphors called "decorators", to obtain a kind of Augmented Virtuality, in order to show either the value of the delay, or the past or the future of a virtual object. Our solutions to provide awareness of network troubles will be inspired from these studies.

3 Managing network troubles

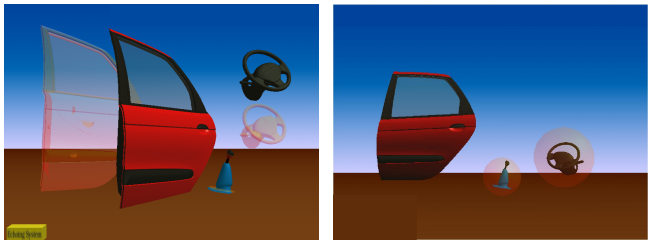
One of the common synchronization methods in distributed systems is the use of periodic synchronization messages. It ensures a hard real-time synchronization between all sites and requires a response to events within a predetermined amount of time to function properly. Network delays or troubles deny events and updates from coming in time, which causes the breaking of the real time concept. In a distributed virtual reality simulation context, the consequence of breaking real time may be one of the two following scenarios: freeze the whole virtual world on all sites until a synchronization message shows up, which is very harmful for the session especially if the delay is important; or let the distributed virtual world goes on even in case of delay or disconnection, which will split up the virtual world to several parallel worlds due to different users interactions on the same virtual object. That is what can be obtained using a Dead-Reckoning algorithm. It can be a good choice if the users do not collaborate closely, but not if several users' interactions are closely coupled. Our work [Duval, T. and El Zammar, C. 2006] is a mix of the two scenarios: we choose to let the simulation continue by freezing up only parts of the world which state is uncertain for consistency considerations. So the virtual world is not out of use while waiting the reception of updates. Even objects interactivity will be preserved for objects that are calculated locally.

3.1 Detection of network delay or disconnection

All the processes involved in an OpenMASK shared simulation send synchronization messages to each other. It allows to coordinate the different processes so that they evolve equally in the virtual world. These messages are used to synchronize an object with his distant mirrors and to carry the updates from a referential to his mirrors. When a site has not received synchronization messages from one site for too long, this site is declared as temporarily disconnected. Each site conserves its own list of disconnected sites. The synchronization message time-out threshold has to be carefully determined according to the characteristics of the network, otherwise it could downgrade the system performance, either by too frequent creation of unnecessary echoes or by detecting the troubles too late.

3.2 The awareness provider system

To visualize differences between a referential and some of its mirrors because of network delay or disconnection, we use an echo object that represents the state of its associated real distant object: the mirror of a referential. An echo has the shape of its associated object but is a little bit smaller and half-transparent so that we can not see it when a simulation is going on normally. When the delay between the referential process and its mirror process is important, we may see a gap between the motion of the referential and the motion of the echo of its mirror (figure 1), and in case of a disconnection, the echo is frozen on the screen and does not evolve any more. This means that the mirror concerned with this echo is not receiving updates any longer because of the disconnection. We use local objects (a kind of referential that can not have mirrors) to create echoes dynamically. Once a site has detected the loss of another distant site, it enables the creation of local echo objects that appear exactly with the last known states (position and orientation for example) of existing referentials. Only one simulation step time separates the physical disconnection of a site from the creation of associated echoes on other sites, so the loss of the last exact value is not really significant. Echoes may also appear with the current state of some mirrors in the scene, since the dynamic echoes creation system detects also mirrors that have "brothers" (mirrors associated to the same referential) located on a disconnected site. This way, users become aware that their interactions with some objects are not perceived any longer by some other users.



(a) (b)

Figure 1: The echoing system (a) and the marker system (b)

[1] www.openmask.org

The marker system surrounds some mirror objects by a half-transparent sphere, which visualizes that these marked objects are not holding the last updated values (figure 1). It marks only the mirrors associated to referentials that exist on a disconnected site. This gives to the user a very specific idea concerning the disconnected sites, and does not charge the scene by undesirable marks.

4 Migration of virtual objects

Beside detecting and visualizing network troubles, we want to provide to users a rescue technique based on virtual objects migration. Some few virtual reality systems like AVIARY [West, A. et al. 1993] and WAVES [Kazman, R. 1996] have implemented object migration to ensure load balancing. Our object migration will ensure a non-interrupted control of a specific object chosen by the user, as on each site, when network breakdowns occur, a user can only control referential objects. Mirror objects lose their interactivity as their evolution depends on the reception of their updates from distant referentials. If a user wants to keep the control of a set of objects, he claims the need of these objects to the migration system that ensures that the user will own these objects locally on his site.

4.1 How do objects migrate ?

We have implemented the object migration system at the kernel level of OpenMASK, it migrates an object by changing the state of its mirror and referential, because this ensures a good continuity of the shared virtual environment. For example, to migrate an object from *site*1 to *site*2 the migration system changes the state of the mirror of the object on *site*2 to make a referential of it, and then it changes the state of the referential on *site*1 to make a mirror of it. If no mirror exists on *site*2, the migration system will first create one. This way there is no destruction of existing objects when they migrate. The different steps to apply in order to make an object migrate (by transformation) to another process are:

1. the computation of the object is temporarily stopped,
2. if it exists, the target mirror is unplugged from its referential,
3. this mirror is transformed into a referential,
4. all the other mirrors are unplugged from the initial referential (their processes unsubscribe to the sending of updates from the process of the initial referential),
5. these mirrors are then plugged to the new referential (their processes subscribe to the sending of updates from the process of the new referential),
6. the initial referential is transformed into a mirror,
7. this last mirror is plugged to the new referential,
8. the computation of the object is enabled.

We have studied two different possibilities allowing to switch on and off between referentials and mirrors. The first solution can be achieved by implementing the migration thanks to an internal mutation of the object. The second solution enables migration by changing the nature of the object. Referentials and mirrors have many common features. Their implementation within OpenMASK is completely transparent to users. The simulated object associated to a referential is the same than the one associated to a mirror except that only referentials are initialized and evolve: their inputs, internal parameters and published parameters can have new values at each simulation step. On the contrary, only the published parameters of a mirror will evolve, to reflect the public values of their associated referential. So, when initializing an object, a user may define a special behavior for this object, and the mirrors may not be informed of all the details of this behavior. The main difference between referentials and mirrors is their associated object manager: a referential object is an association between a simulated object and a referential manager, and a mirror object is an association between a simulated object and a mirror manager.

The first method consists in reconstructing the internal structure of referentials and mirrors so that they implement exactly the same interface. This means that a referential will contain the same functionality than a mirror, in other terms it is considered as a referential and mirror at the same time. Referential or mirror interface will be enabled or disabled accordingly to the need of the migration system. The advantage of this method is that the mutation of an object from a referential into a mirror is very easy to realize at run-time and conversely. The disadvantage, mainly because of our OpenMASK context, is the important structure modifications that may change deeply the software architecture of a simulated object. Moreover this will generate a heavy object structure that contains referential interface and mirror interface at the same time.

The second method does not change the structure neither the interface of referentials and mirrors. Migrating an object by changing its nature consists in removing the manager of an object and replacing it without destroying the object itself. For example, to transform a mirror into a referential, we only destroy its mirror manager and we replace it by a referential manager. In case a user has given some specific behavior to an object, he can migrate this behavior by redefining two extra methods *emigrate()* and *immigrate()*, which are kinds of serialization and deserialization methods that allow to determine all the important information to transmit from the old referential to the new one, within a dedicated message.

Although the first method would probably have been the more efficient one, we adopted the second method to implement object migration within OpenMASK, because the first method would have imposed too much modifications within the OpenMASK kernel.

4.2 Migration protocol

A special event is sent to start the migration of an object. It contains the name of the object and the name of the destination process that may be on a distant site. On each process, the local controller has to process this event accordingly to one of the following cases:

- the controller owns the referential of the migrating object. In this case, this referential must be transformed into a mirror by destroying its referential manager and creating a mirror manager. The controller will transmit to this mirror manager the name of the process where the referential is located.
- the controller is located at the destination process and it controls a mirror of the migrating object. In this case, this mirror must be transformed into a referential by destroying its mirror manager and creating a referential manager instead. The controller will record that from now on, it will have to send updates to all the mirrors of its new referential.
- the controller is located at the destination process but it does not have a mirror of the migrating object. In this case, a mirror should be created. Then, it will be transformed into a referential as explained in the previous case.
- the controller has a mirror copy of the migrating object without being located on the destination process. In this case, this mirror should be directed towards the new referential so that it can go on receiving updates.

As previously detailed, the migration steps must be done in one particular order, so all the local controllers will not receive these different migration events at the same simulation step. As some of the

simulation steps can be parallelized, our OpenMASK implementation enables a migration in two simulation steps, which is quite a good optimization of the migration process.

4.3 Evaluation of the migration mechanism

We have just explained that the migration of an object to another process is carried out in only two OpenMASK's simulation steps, which duration is around 13ms for a 75Hz simulation. To evaluate our migration mechanism, we realized an experiment to measure its efficiency in term of visual perception, *i.e.* to answer this question: can a user perceive the migration of an object ?

The application we have realized for the experiment contains 100 moving objects with a 75Hz frequency. These objects are following each other, and the first object is following an arbitrary trajectory. The objects are associated to two processes *A* and *B* distributed on two different computers. All referential objects were on process *A* at the beginning of the simulation, and the experiment consisted in making some objects migrate from process *A* to process *B* and to study the visual impact of the migration while increasing progressively the number of migrating objects. The measured results of this experiment show that we need around 13ms + n*0.25 ms to migrate n objects. The computers we used were Pentium IV activated at a 3 GHz frequency, connected by a gigabyte local network.

From the visual perception point of view, we can observe that the application freezes for a very short moment while the migration is on, this is true whatever the number of migrating objects, as the lower cost of migration is about 13ms. This 13 ms threshold could be reduced thanks to one particularity of our OpenMASK Kernel that allows each simulated object to have its own activation frequency. It is possible to give a higher activation frequency to the OpenMASK controllers (for example 150 Hz, 300 Hz or even 600 Hz), but it would also have its cost (as the controllers would be activated at a higher frequency, there would be an increase of computation weight) and some experiments have to be made to find what could be the optimal frequency value for a given set of computers.

5 Conclusion

We provide a virtual object migration mechanism, which is used to ensure a non interrupted control and manipulation of an object by migrating this object to the local site of a user, since local interactions are not sensitive to a network problem. This mechanism is coupled with a synchronization message time-out detection mechanism and two kinds of metaphors to inform users about the availability of the updates of an object (echoes and marks).

Our contributions allow users to go on working even when network troubles appear because they can become aware of the problem as soon as the OpenMASK kernel detects the increase of the delay. So they can temporarily delay closely coupled cooperative interactions, to focus upon interaction they can manage locally, may be with objects that can migrate from one process to another. They can also go on with cooperative interactions with users whose sites are not affected by the troubles. As soon as the network problems disappear, the users can come back to their previous collaborative tasks. The migration is possible only if the network is not totally down, so if a user wants to make some objects migrate because of network troubles, we hope that he could have been aware of the problem thanks to the echoing and marking systems, which is possible only if the network does not breakdown too suddenly and with a good synchronization message time-out threshold.

Virtual objects migration allows several kinds of extensions such as the dynamic management of processes and users by adding or removing sites to an already running simulation. Another possible extension is the dynamic management of areas of interest by migrating automatically a set of objects to a particular site depending on the interest of users, for example depending on the 3D position of the user, to ensure that most of the objects he can interact with are located on his site.

References

BOWMAN. D., KRUIJFF E., LAVIOLA J., AND POUPYREV I. 2004. *3D User Interfaces: Theory and Practice*. Addison-Wesley Eds.

DUMAS, C., DEGRANDE, S., SAUGIS, G., CHAILLOU, C., VIAUD, M., AND PLENACOSTE, P. 1999. SpIn: a 3D Interface for Cooperative Work. *Virtual Reality Society Journal*.

DUVAL, T., AND EL ZAMMAR, C. 2006. Managing Network Troubles while Interacting within Collaborative Virtual Environments. *CSAC'2006, Paphos, Cyprus*.

DUVAL, T., AND LE TENIER, C. 2004. Interactions 3D coopératives sur des objets techniques avec OpenMASK. *Mécaniques et Industries 5*, 2, 129–137.

FRASER, M., BENFORD, S., HINDMARCH, J., AND HEATH, C. 1999. Supporting Awareness and Interaction through Collaborative Virtual Interfaces. *UIST'99, Asheville, USA*, 27–36.

FRASER, M., GLOVER, T., VAGHI, I., BENFORD, S., GREENHALGH, C., HINDMARCH, J., AND HEATH, C. 2000. Revealing the Realities of Collaborative Virtual Reality. *CVE'2000, San Francisco*, 29–37.

GUTWIN, C., AND GREENBERG, S. 1998. Design for Individuals, Design for Groups: Tradeoffs Between Power and Workspace Awareness. *CSCW'98, Seattle, Washington, US*, 207–216.

GUTWIN, C., BENFORD, S., DYCK, J., FRASER, M., VAGHI, I., AND GREENHALGH, C. 2004. Revealing Delay in Collaborative Environments. *CHI 2004*, 503–510.

GUTWIN, C. 2001. Effects of Network Delay on Group Work in Shared Workspaces. *Proc. ECSCW*, 299–318.

KAZMAN, R. 1996. Load Balancing, Latency Management and Separation of Concerns in a Distributed Virtual World. *Parallel Computations - Paradigms and Applications*.

MACEDONIA, M., ZYDA, M., PRATT, D., BARHAM, P., AND ZESWITZ, S. 1994. NPSNET: a Network Software Architecture for Large Scale Virtual Environments. *Presence, Vol3, No.4*, 265–287.

MARGERY, D., ARNALDI, B., CHAUFFAUT, A., DONIKIAN, S., AND DUVAL, T. 2002. OpenMASK: Multi-Threaded or Modular Animation and Simulation Kernel or Kit : a General Introduction. *VRIC 2002 Proceedings*.

PARK, K., AND KENYON, R. 1999. Effects of Network Characteristics on Human Performance in the Collaborative Virtual Environment. *Proc. of IEEE Virtual Reality'99, Bonn*, 104–111.

VAGHI, I., GREENHALGH, C., AND BENFORD, S. 1999. Coping with Inconsistency due to Network Delays in Collaborative Virtual Environments. *Proceedings of the ACM symposium on Virtual reality software and technology*, 42–49.

WEST, A., HOWARD, T., HUBBOLD, R., MURTA, A., SNOWDON, D., AND BUTLER, D. 1993. AVIARY - A Generic Virtual Reality Interface for Real Applications. *Virtual Reality Systems (sponsored by the BCS), May 1992*, 213–236.