Virtual world objects for real-time cooperative design

Christian Toinard[1] and Nicolas Chevassus[2]

[1] CEDRIC, Centre d'Etudes et de Recherche en Informatique CNAM, 292 rue Saint-Martin, 75141 Paris Cedex 03, France toinard@cnam.fr
[2] AEROSPATIALE - Centre Commun de Recherches,12 rue Pasteur, BP 76, 92152 Suresnes Cedex, France NICOLAS.CHEVASSUS@siege.aerospatiale.fr

Abstract. This proposition presents a system architecture for the "rapid prototyping" of manufacturing products. This architecture satisfies the following requirements: cooperative design of a shared virtual world, concurrent and real time interactions of different users, dynamic distribution of the virtual scene over the different users, distributed world consistence, persistence and faults recovery. Currently, few solutions answer entirely these requirements. The virtual scene is dynamically distributed over private spaces according to the cooperative interactions. A user is aware in real time of the operations carried out by the other participants. A distributed concurrency control allows consistent operations on the distributed scene tree. The persistence is provided in a distributed way. A user can go on working despite the faults of other machines.

1 Presentation and justification

Principles: the system manages a copy of the scene for each user. A copy is composed of two parts: a private space and replicas of distant private spaces.

A private space enables a participant to work, at the same time, on multiple subtrees of the scene. Concurrent operations are processed within different private spaces. A subtree can be transmitted to another private space. A private space is consistent because it is modified by local interactions. A replica can be inconsistent but the inconsistency can be recovered through a subtree transmission. A private space is persistent through savings in the user home directory. The private spaces and the replicas can run without any central server. As explained latter, only a subtree transmission requires a central server. Moreover, a private space can be used on an isolation basis: it does not require the presence of any distant machine.

A public space is involved in a subtree transmission. The public space runs on a central server. That machines provides name services in order to localize a subtree. A user can update the public space with saving operations when he decides to make more available his private space modifications. Thus, the location of a newly created node becomes available to the distant users. Otherwise, a new node can be observed by a distant user but can not be transmitted to the distant private space. The public space realizes a redundant storage (in the event of breakdown of a private space). The public space does not ensure the global persistence. In fact, the persistence is carried out in a distributed way by the various private spaces.

Generally, the solutions do not distribute the scene tree according to the cooperative interactions and consider one space of work. Solutions [12] [8] [1] [11] manage mechanisms to observe only part of the space and to thus filter the received events. These solutions rest on a cutting of the virtual world in cells in order to limit the size of the presented graphic scene. The world is divided according to a geographic or static cutting. [8] defines a dynamic cutting (which is not perfect) according to the motion of objects. [5] [11] do not define how to make consistent the copies of the scene with respect to concurrent actions of the users. [1] considers a technique of concurrency control to manage the motion of objects.

Our system specifically addresses the cooperation. The animation of moving objects in a simulation context (video games of military simulations of battle fields) is not our main purpose. A distributed cooperation on a global scene is provided with disjoined private spaces. A private space subtree can be split again into several subtrees in response to other users. Thus, the distribution is processed dynamically according to the user requests. Different cooperation schemes are allowed using a distributed concurrency control for the distributed scene tree. Thus, concurrency is achieved while preserving consistency. This proposition is original and fits a cooperative activity well.

Real time animation: A user operation (creation, destruction, update of a node) is transmitted in real time to the replicas. The events are multicast through a best effort transmission using UDP. The solution does not require any reliable multicast. Thus, the transmission time is as short as possible.

As no technique of reliable transmission nor of scheduling is used, a replica can be inconsistent. The inconsistency is recovered when required using the subtree transmission. For example, a user whises to modify a subtree that belongs to a distant private space. Then, the subtree transmission is processed and all the possible inconsistencies are recovered for the requesting user. At the same time, the subtree transmission allows the error recovery and the consistency.

Generally, the solutions of the literature aim at a real time which is as much as possible exact [4] [8] [12] [5] [11] [1]. Some [5] [9] [1] try to recover the transmission errors. For the update of moving objects, solutions [5] [9] try to transmit a recent state of the object at the time of a transmission error. Our solution uses the private space transfer to recover the errors when it is necessary for the work consistency. The simplicity has a drawback since the quality of animation is not perfect.

Distributed designation: the system provides a distributed designation of the scene objects and nodes. At the root of the tree a unique name is attributed locally when a user creates a new object. The unique name is composed of the address of the creation machine plus a local date (e.g. name @IP1,date1 defines an object subtree). Subsequent nodes are designated according to their position in the subtree of the corresponding object (e.g. @IP1,date1,2 is the second node at the first level of the corresponding object subtree). Only the user that is the owner of a node (this node belongs to his private space) can create a child node (e.g. the first child name @IP1,date1,2.1 is created by the private space that includes the father node @IP1,date1,2). That way, unique names are defined in a distributed way by the different private spaces.

Some solutions define a distributed designation of cells like [1]. In that case, the cell is managed by a unique server. The cell is not able to move to another server because the name is associated with a classical URL (Uniform Ressource Locator). Others like [5], show how to localize the nearest copy for a given object. For us, at a given time a single copy (private space) replies to a name. This name can move from one copy to another one. A global server allows to localize the moving name. This name server is used when a subtree transmission is requested. Its breakdown does not prevent the system from running. It prohibits subtree transfers. As the names are generated locally, the name server is not involved by the creation of a new object or a child node (when the father already belongs to the private space).

Concurrency control: The name server, managing the shared space, permanently maintains the ownership of a subtree. A copy, which requests the transfer of a subtree, sends an ownership request to the name server. This one replies with the identity of the current owner and memorizes the identity of the requester. Then, the subtree transmission is achieved between the new owner and the requester. This distributed concurrency control allows different ways of working. A participant can run on an ownership basis and preserve the ownership until he accepts modifications. He can run on a community basis and accepts at any time to let another user have the ownership. Other kinds of cooperation are allowed.

In [1], a mechanism is proposed to acquire a lock in order to solve the conflicts on the motions of the objects. In particular, a lock is released automatically on time-out so that motions are not blocked. In our case, a distributed concurrency control is processed within a distributed scene tree. It is used to carry out a consistent cooperation. It specifically does not relate to motion operations (update). Several way of working are allowed (ownership, community, etc).

Persistence The solutions of persistence [3] [1] generally use a centralized server who ensures the persistence of the scene. In our proposal, the persistence of a private space is not dependent on any centralized machine. The consistent operations are persistent through local savings. The owner can work locally without requiring the presence of the name server. The shared server makes available a work at a certain stage. It also realizes a redundant storage. Thus, the persistence is carried out in a distributed way and a certain level of fault tolerance is provided.

2 Conclusion and future work

This paper presents the principles of system services to support real time cooperative design within a virtual environment. A real time awareness between the different users is provided. The graphic scene is distributed in order to achieve concurrency and cooperation. A dynamic distribution is provided. A distributed concurrency control allows to preserve the consistency of the distributed scene. A practical solution is proposed to manage the faults of different machines. Currently, we are working on the implementation and the integration of this system in the framework of cooperative design application.

References

1. Broll, W.: Distributed Virtual Reality for Everyone a Framework for Networked VR on the Internet. In Proc. of IEEE 1997 Annual International Symposium on Virtual Reality. Albukerque, USA, 1-5 May 1997.
2. Hussein, K., Pea-Mora, F., Skiram, R.D.: CAIRO: A system for Facilitating Communication in a Distributed Collaborative Engineering Environment. Fourth IEEE Workshop on Enabling Technologies Infrastruture for Collaborative Enterprise, April (1995).
3. Leigh, J., Johnson A.E., Vasilakis, C.A., Defanti, T.A.: Multi-perspective Collaborative Design in Persistent Networked Virtual Environments. http://jaka.eecs.uic.edu/ spiff/calvin.
4. IEEE Computer Society: IEEE Standard for Distributed Interactive Simulation Applications Protocols. IEEE Std 1278.1 (1995).
5. Hagsand, O.: Interactive Multiuser VEs in the DIVE System. IEEE Multimedia, **3-1** Spring (1996).
6. Dourish, P.: Consistency Guarantees: Exploiting Application Semantics for Consistency Management in a Collaboration Toolkit. In Proc. ACM Conference on Computer-Supported Cooperative Work CSCW'96, Boston Mass. ACM (1996).
7. Florin, G., Toinard, C.: A new way to design causally and totally ordered multicast protocols. ACM Operating Systems Review, **26-4**. October (1992).
8. DIS++/HLA Frequently Asked Question. http://www.dmso.mil. 22 April (1996).
9. Kessler, G.D., Hodges, L.F.: A Network Communication Protocol for Distributed Virtual Environment Systems. Virtual Reality Annual Symposium , Santa Clara, CA(1996).
10. Kindberg, T.: Mushroom: a framework for collaborative and interaction across the Internet. Proc. CSCW and the Web. 5th ERCIM/W4G workshop (1996).
11. Barrus, J. W., Waters, C., Anderson, D.B.: Locales: Supporting Large Multiuser Virtual Environments. IEEE Computer Graphics and Applications November 1996.
12. Macedonia, M. R. et al.: Exploiting Reality with Multicast Groups. IEEE CG & A **15-5** (1995).
13. Stytz, M.R, Adams, T. Garcia, B., Sheasby, S.M., Zurita, B.: Rapid Prototyping for Distributed Virtual Environments. IEEE Software (1997).
14. TASC, Inc.: A distributed Interactive Simulation Intranet Using RAMP, a Reliable Adaptative Multicast Protocol. http://www.tasc.com.
15. Toinard, C., Chevassus, N.: An object communication architecture for real time distributed process control. ECOOP97 Workshop on OO Technology and Real Time Systems. Jyvaskyla Finland (1997).
16. Toinard, C., Chevassus, N.: CORBA for distributed and fault tolerant process control. ECOOP97 Workshop CORBA Implementation Use and Evaluation. Jyvaskyla Finland (1997).