Collaborative design using distributed virtual reality over the Internet*

Fabien Costantini**[a], Christian Toinard**[a], Nicolas Chevassus[b], François Gaillard[b]
[a]CNAM-CEDRIC, 292 rue Saint-Martin 75141 Paris Cedex 03 France
[b]EADS Corporate Research Laboratory, 12 rue Pasteur BP 76 92152 Suresnes Cedex France

ABSTRACT

Efficient collaborative virtual environments are missing. First, current solutions do not support mobility to move easily from a disconnected work to a meeting. Second, they do not preserve the consistency or they limit the parallel working. Third, a client-server approach is inefficient in many ways. It introduces a bottleneck and a point of failure in the system. At last, requiring a specific Quality of Service (QoS) from the under-laying network limits the ease of deployment. Paper answers these shortages. It enables to distribute a global scene tree among several private spaces. A worker carries out a disconnected work to improve his private space while satisfying protection rules. These different puzzle pieces assemble automatically into a global scene tree during a meeting work. Workers modify in real-time the shared scene. Real time awareness, parallel working and work persistency are provided. A consistency property guaranties the work progression. The solution is fully distributed. Full replication and multicasting improve the performances. A connection facility solves the connectivity problem. It switches automatically from multicast to point-to-point. At last, security is addressed. Standard and secure email enables authentication and distribution of a session key. A re-keying protocol assures confidentiality without requiring communication entities to process X509 certificates.

Keywords: Distribution services, collaborative design, distributed virtual reality, Internet, security

1. INTRODUCTION

Several solutions are proposed in the context of collaborative virtual design. They come from virtual prototyping, distributed virtual reality or computer supported collaborative work (CSCW).

First, virtual prototyping products provide end-user solutions using virtual reality for design review[6.,13.,19.], collaborative design[19.] or the data management[12.]. These solutions support synchronous work[6.,13.,19.] or asynchronous work[12.]. In essence, these solutions use a central database approach and a server to support synchronous work. The main purpose is collaborative design review. Despite extensions with real-time creation of the shared prototype[12.], these solutions address poorly the parallel and mobile working since they do not manage distribution and protection of a virtual scene tree. As end-user products, these solutions do not offer collaboration services that any application could use to support collaboration facilities.

Second, distributed virtual environments provide an application programming interface (API) to share a virtual scene. Historically, distributed virtual reality comes from battlefield simulations. Their major intent is to reduce the network traffic due to a high number of moving objects. Replication and multicasting improve the performances. A cutting of the scene also improve the performances using different multicast addresses through a dynamic area-of-interest[3.,9.] or a static decomposition[2.,4.]. A consistent progression of the work is not guarantied since neither the reliable transmission[4.,9.] neither the locking mechanisms[4.,16.,53.] nor the logical ordering[9.] preserve the work carried out by concurrent users. Since a scene tree cannot be distributed among different private spaces for disconnected improvements, mobile working is poorly supported. All these solutions present a server for the persistency[16.,53.], management of participants[4.,53.], error recovery[4.] or event distribution[53.]. The multicast connectivity is not addressed[2.,3.,4.,16.]. A time synchronization[4.] or a resource reservation[9.] limits the usability of the solution.

Third, CSCW environments provide either the basic programming toolkits[1.,26.,28.,41.,43.,46.] or the end products[17.,35.,47.]. They enable to share documents and resources among a group of workers. The end products[35.] can use standardized solutions[24.] to support a real-time transmission of audio/video streams using the Real Time Protocol[45.]. Sharing[1.,17.,26.,28.,35.,41.,43.,46.,47.]

* Work supported by the European Commission through the IST-AIT VEPOP project under contract number IST-1999-13346, EADS Corporate Research Lab, EADS Airbus, CNAM-CEDRIC, Flow-Master, University of Oulu
** {costanti,toinard}@cnam.fr; http://cedric.cnam.fr; Tel/Fax:+33 0140272296

enables to distribute events/data among a group of replicas. Thus, each replica maintains a copy of shared resources. Either a central server[1.,26.,28.,41.,43.,46.] with Transmission Control Protocol[39.] (TCP) connections or a reliable multicast[26.] is used to recover from transmission errors. But, reliable transmission provides a poor real-time awareness. A consistent progression of the work is not guarantied[1.,17.,26.,28.,41.,43.,46.,47.] or a parallel working[35.] is not supported. These solutions permit neither to distribute a virtual scene tree among disconnected workspaces nor to reform automatically the global scene.

So, current solutions do not address efficiently collaborative virtual prototyping. First, they do not support mobile working since they are not capable to distribute a scene tree among private spaces for disconnected improvements and automatic aggregation. Second, they do not guaranty a consistent progression of the work. Third, they provide bad performances and a low availability if they use a central server. Fourth, when using multicasting, they do not address the multicast connectivity problem. Fifth, requiring a specific quality of service like resource reservation or time synchronization is not an open solution since these service are not always available at any workstation. At last, none of them provide security in the context of un-secure Internet channel. This problem is really opened when using UDP[38.] to multicast messages since there is no standardized and widely available solution for UDP transmission. Security at the network layer[22.] is not satisfying since every Internet network does not support it. In the context of H.323[24.], security is proposed through the H.235[25.] standard. But, it assumes the ability for communication entities to process X509[23.] certificates.

2. DISTRIBUTED SERVICES

2.1. Framework Principles

One proposes a framework that is a collection of distributed services. It supports two main types of working.
Meeting enables to share a global scene tree and achieve real-time modifications (i.e. creation, deletion and update of node). The meeting assembles automatically the different pieces of the scene puzzle. For that purpose, each participant brings his private space in the meeting. Each entering space contains different pieces of the global puzzle. The overlapping of the different private spaces is correctly managed. In that case, it is guarantied that the correct piece takes the right place. When leaving a meeting, a participant defines the pieces of the puzzle that he wants in his private space. In fact, each node of the scene can be viewed as a piece of the global puzzle. When leaving, any participant selects independently the different nodes of the puzzle. So, the private spaces overlap with each other when different participants select the same nodes.
Disconnected work enables a designer to improve the global scene without requiring any network connection. He can add new pieces to the global puzzle or delete or update pieces of his workspace. Despite a disconnection, created (deleted) pieces increase (decrease) the size of the puzzle. During a further meeting, the global puzzle will reform by assembling automatically the different pieces at the right place.

The use of the global scene tree must respect the rights of the different participants. For that purpose, each node has an owner. The owner defines the read/write rights that are provided to the other workers. Ownership and rights evolve dynamically all along the node life. Thus, ownership can be transmitted to a distant worker during a meeting. Read and write attributes can be changed to extend or restrict permissions. The system guaranties that there is a unique owner at a time. This property is satisfying since generally a unique designer needs to process a node. Moreover, this property permits to resolve overlapping private spaces since the owner has a consistent node.

The scene puzzle offers a fine grain schema. It authorizes a high level of parallel working since two distinct nodes are processed simultaneously. Parallel working is fully supported both during disconnected work and during meeting. Despite a high level of parallelism, a consistent progression of the work is guarantied. Any operation occurs starting from the latest state of the corresponding node. Thus, the system prevents a participant from redoing his work because of a concurrent operation.

During a meeting, each participant gets a replica of the global scene. Each interaction is processed locally and sent to the distant replicas. Each replica processes the received interaction as if it was originated locally. Thus, a replica updates all the distant replicas through message passing. The message contains all the interaction attributes (type, node reference, node value, ownership, rights).

In the following section, the different services and qualities of distribution are described. The motivation of each service or property is first given through a description of the addressed problem.

2.2. Distributed Designation

2.2.1. Problem

The nodes of the global scene must keep unique names in time and space among the different private spaces and replicas. Figure 1 shows a scene tree related to a ventilation system. That scene tree includes an electrical sub-system and hydraulic sub-system. Figure 1 presents also the 3D rendering of that scene.

The solution must guaranty several properties.
First, unique names must be easy to manage and use.
Second, two private spaces or replicas cannot contain the same object with two distinct references. During a disconnected phase, the pump can be present in two different private spaces and still be designated with the same name. During a meeting, two replicas use a same name to access the pump object (local or distant).
Third, if a name designates two distinct nodes then uniquely one node is active and the other is a pending node that cannot be processed. The pending node must disappear and be replaced by the active one. The system must guaranty that a pending node exists only within a private space. In contrast, two versions of the same node have the same name.

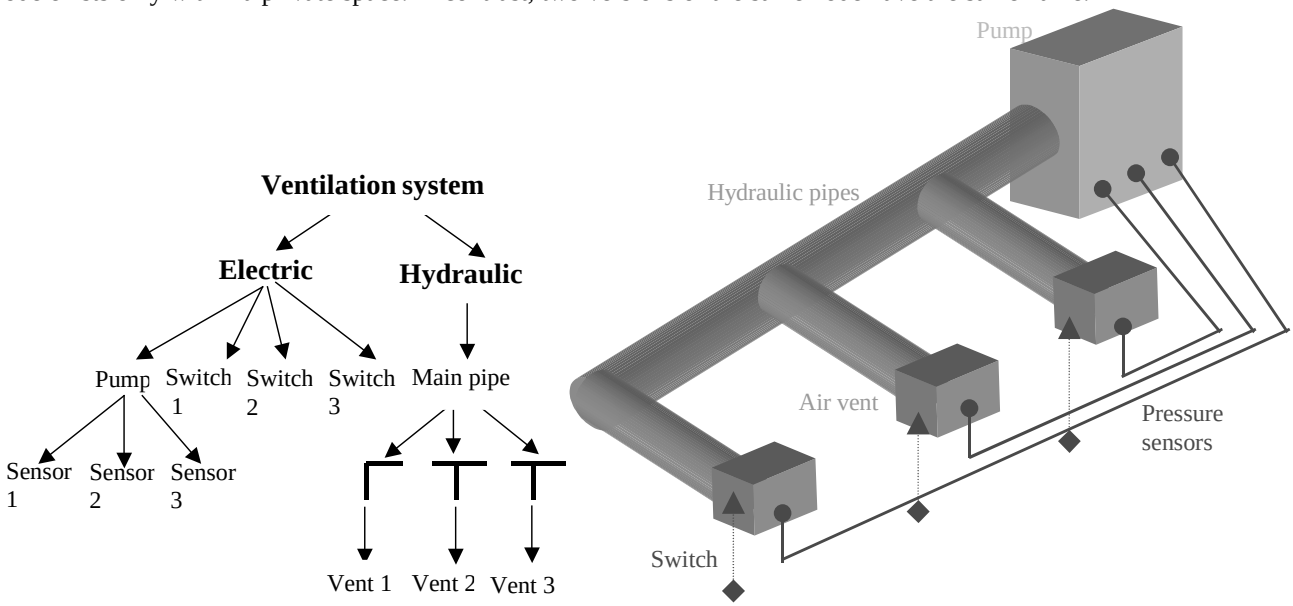

Fig. 1 Scene example

2.2.2. Solution

A peer (i.e. an entity processing a replica or a private space) computes locally a unique name when a user creates a new node within the scene tree. A unique name contains the Internet Protocol (*IP*) address (*@IP*) of the creation machine, a local timestamp and the position within the tree (e.g. *@IPA,stampA,1.3* corresponds to the first child and third grandson of the node *@IPA,stampA*). A timestamp includes a date, corresponding to the local creation date, plus a random number. Moreover, a node maintains locally the child names. For example, node *@IPB,stampB,1* stores locally the list of its children (*@IPB,stampB,1.1*, *@IPB,stampB,1.2*, *@IPB,stampB,1.3*). To create or to delete a name, a user must be the owner of the father. For example, to create name *@IPB stampB,1.3* the user must own node *@IPB,stampB,1*.
Thus, a node (e.g. *@IPA,stampA*) can be created at the first level on a disconnected basis without being attached to any particular scene. It enables a user to prepare some work without knowing in advance in which scene he will enter his work. For example, one designer prepares an electric sub-system and another designer prepares an hydraulic sub-system. Afterwards, they reach a meeting to design a ventilation system and bring their works. Figure 2 gives the automatic merging of the corresponding scene tree. The root of tree corresponds to the name of meeting. It is seen that the unique names do not correspond directly to the graphical position of the objects (e.g. the first vent is named *@IPB,stampB,1,3* while the second vent is named *@IPB,stampB,1,1*).

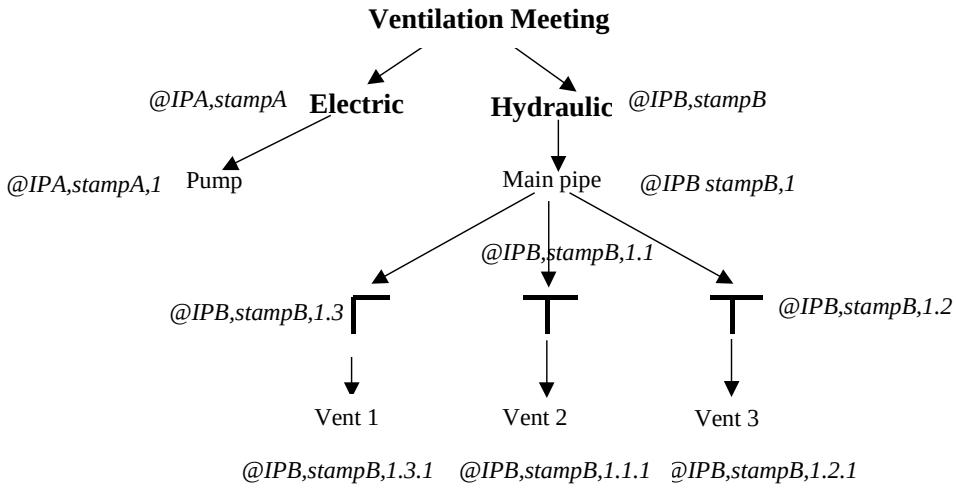

Fig. 2 Distributed unique names

The first property is satisfied. The unique names are easy to create since each name is created locally. Thus, a unique name can be created both during disconnected work and during meeting. A unique name is independent of the geographical position within the 3D world. So, objects can move without changing their names. A unique name defines the position of the node within the global scene tree. Thus, a name is unique, in time and space, but defines also a position within the scene tree. It is useful since the application retrieves directly from a name which node is the father.

The second property is satisfied since the object name is defined at creation time and remains unchanged all along the life of that object.

Third property is guarantied because to create a name (*@IPB,stampB,1.3*) a user must be owner of the father (*@IPB,stampB,1*). So, the father must be present to create a child node. Using the local list of children, the father guaranties that an active name cannot be reused. To be reused, the owner of the active name must be present and must delete the name. If the same name exists for another object, it can be only for a user that is neither present nor owner. So, it can only be a private space that contains the pending name. Moreover, the pending name will disappear when the user will reach a new meeting.

It should be noted that a peer can change its *IP* address without any difficulty. An already created name remains unique in time and space since a name is unchanged all along the node life. The new *IP* address will produce distinct names because the probability that two machines use the same address to produce the same name (same local time and same random number) is close to zero. If another machine allocates (in the future) the same *IP* address with completely de-synchronized clocks, the random numbers will discriminate the two names.

2.3. Distributed Protection

2.3.1. Problem

Despite a distributed processing, the system must satisfy a unique protection rule while preserving the distributed designation.

First, an operation must preserve the protection of the concerned node. For example, a participant needs a write permission to modify the pump color.

Second, a unique owner exists between the different private spaces and replicas. Thus, two distinct private spaces (distinct replicas) cannot present two different owners for the pump.

Third, a unique protection rule exists for one node version among the different private spaces and replicas.

Fourth, a user cannot write a node in his private space if another copy does not enable that writing. Thus, unauthorized actions are prevented among disconnected private spaces.

Fifth, the protection must enable to respect the distributed designation rules. Let us give a short example of problem that must be avoided. A user is the owner of a name, designating the pump, and forbids any distant operation on the pump. As owner of the electrical sub-system, a second user deletes the electric node. Afterwards, that second user creates a new electrical sub-system reusing the deleted name. At last, he creates a second pump from the newly created electrical sub-system. Thus, parent rights could permit to define the same name for a second object. So, protection rules must efficiently contribute to the management of the unique names.

2.3.2. Solution

To have a read permission for a given node, the read attribute must be set for that node. Write permission is given if the user is the node owner. A distant user can request a write permission when the write attribute is set. In response, he acquires the ownership for the requested node. The protection attributes (read/write) are transmitted within each creation, deletion and update event. Thus, any distant user modifies the protection according to the rights defined by the producing owner. To create a node, the user must own the father node. To delete a node, the user must own the father node and children nodes.

First property is guarantied since the system checks the required permission before processing any operation.
Second property is satisfied because ownership is requested to the current owner. The current owner grants the ownership to the requester. So, there is a unique owner at a given time.
Third property is guarantied. If two private spaces present two different protection rules it is for two different versions of the corresponding node. Let us explain by contradiction. Assume two copies with the same version and different protection rules. These two copies recover the same object version from the same owner. So, they get the same protection rule from that owner. It is conflicting with the assumption.
If a user has write permission for a node within his private space, then he is the owner. Since there is a unique owner at a time, no other user can have the write permission. So, the fourth property is satisfied.
A user must own the father to be able to create a node. Father ownership prevents the father from reusing the name of an active node. Thus, the newly created node has a unique name. A user must own the father and children nodes to be able to delete a node. Generally, one cannot delete a node without deleting previously the corresponding sub-tree. So, requesting ownership of the children is used to prevent deleting non-terminal nodes. Children ownership prevents also the children from deletion without informing their father. At last, children ownership prevents the creation of a new child while the father disappears. Father ownership prevents the father from reusing an active name before it is destroyed. Thus, the system guaranties that unique names exist while reflecting a correct position within the scene tree. It must be noted that the solution does not require the sub-tree ownership for the concerned node. So, the overhead of a node creation/deletion is limited to parents.

2.4. Meeting Schedule

2.4.1. Problem

The users must be able to schedule a meeting. Several participants must be able to find a satisfying date and hour for the meeting. Relevant information, such as communication channel and session key for the meeting, must be transmitted in a secure way. Thus, distant participants must be authenticated and confidentiality is guarantied for the relevant information.

2.4.2. Solution

The solution consists in using standard email clients (like Microsoft Outlook or Netscape Messenger) to negotiate the meeting. S/MIME transmits securely the communication channel and session key.

It provides the following services:
- Negotiation: project members use email to schedule a meeting. A member sends an email to the group and starts the meeting negotiation. Each member replies to the group. Several email exchanges can be used before finding a convenient schedule. At the end of that negotiation process, each member authenticated the distant participants. Each member recovered the X509 certificates from the different participants. Thus, confidentiality can be guarantied during further exchanges using the received X509 certificates. Further confidential S/MIME transmissions are used for address allocation and distribution of a session key.
- Address allocation: after the negotiation, a multicast address is allocated using different email messages. In practice, each recipient can use a local directory service (i.e. Lightweight Directory Access Protocol [Wah97]) to reserve the requested address. But, this reservation procedure can be omitted and a conflicting address will be detected and resolved automatically during the meeting.
- Distribution of a session key: the meeting initiator allocates a private key PK_S (i.e. a DES key) that will be used for the symmetric encryption of the meeting. He sends a confidential email to distribute the private key PK_S. That private key is transmitted securely using S/MIME. Thus, only authenticated participants can decrypt the session key PK_S.

2.5. Real Time Meeting Membership

2.5.1. Problem

Each participant must be able to reach a scheduled meeting and know who is present. Thus, each participant has the knowledge of the meeting membership. Moreover, a color must be associated to each meeting member. This is a user-friendly service allowing to color the scene objects according to their owner.

2.5.2. Solution

An entering peer multicasts the participant name to the allocated multicast address. Distant participants reply as multicasting their names. Thus, a new participant maintains locally his knowledge of the meeting membership.
Afterwards, a protocol allocates a color to each participant. Let us describe the protocol. The entering peer multicasts a request including its network extremity (IP address + port number). In response, the distant peers multicast their name, network extremity and color. To terminate the protocol, the entering participant multicasts a second message with the chosen color. When two peers are conflicting for the same color, the peer with the smallest extremity changes its color and multicasts a new message. Thus, each participant gets a color and knows the distant colors. The protocol can have minor effects like seeing one user changing his color during the same meeting.

2.6. Automatic Scene Aggregation

2.6.1. Problem

After the membership phase, a participant must be able to bring pieces of the scene puzzle and gets a copy of the global scene. The different pieces must go to the right place of the puzzle. For each arrival, the distant participants must observe the added pieces. Moreover, each peer must recover from transmission errors and gets a fresh state for each node of the global scene.

2.6.2. Solution

An entering peer multicasts a list part [*LIST*: V_L, *TotParts*, *NPart*, (G_1, V_1),…, (G_N, V_N)] where V_L is the version number of the list, *TotParts* is the total number of parts forming that list, *NPart* is the part number and each couple (G_X, V_X) defines the unique name and the version number for an object *X*. Using *TotParts* parts, a peer announces the objects it enters into the meeting. Afterwards, the announcing peer multicasts an object state [*State*: V_L, G_X, V_X, T_X, S_X] for each object *X* of the list where T_X is the type of *X* and S_X is the state associated with V_X. Thus, a distant peer recovers all of the announced objects. At the end of the scene aggregation, each peer has a copy of the shared scene.
Since a distant peer has the list of the objects, it can ask the retransmission of a missing object *X* by requesting a version number equal or higher to V_X. Thus, a producing peer has to retransmit uniquely the current version V'_X of *X* ($V'_X \geq V_X$).
Since each state [*State*: V_L, G_X, V_X, T_X, S_X] transports a list version V_L, a receiving peer that misses the corresponding list message can ask a retransmission by requesting the list part for the object G_X. Generally, the producing peer will retransmit the corresponding list part [*LIST*: V_L, *TotParts*, *NPart*, (G_1, V_1), …, (G_X, V_X), …, (G_N, V_N)]. When the producer added or deleted an object, the requested version V_L is no more available since the list changed. In that latter case, the producer retransmits all the new parts [*LIST*: V'_L, …] with $V'_L \geq V_L$. After receiving the new list, the requester has the responsibility of asking the retransmission of missing states based upon the received list.
It should be noted that a selective list [*SLIST*: V_L, T_L, *TotParts*, *NPart*, (G_1, V_1),…, (G_N, V_N)] transmits only object references of type T_L. A selective list is a subset of the list V_L.

Each participant brings pieces of the scene puzzle using the list messages. A couple (G_X, V_X) transmits a piece *X*. The entering participant gets a reply from distant participants. Thus, he completes the global scene as assembly of the different responses.
Each piece X goes to the right place since the name G_X contains the location within the scene tree.
Each time a participant enters, the distant copies of the global scene are updated using the list messages. Moreover, the responses give an opportunity for the copies to resynchronize and to recover fresher states. When a receiving peer detects a transmission error (missing announced objects or missing list parts), he asks for newer versions. Thus, retransmission only gives the fresh version of the missing object or missing list.

2.7. Real Time Awareness

2.7.1. Problem

A participant must be aware of the operations carried out by a distant participant in a short time. The solution must not rely on a resource reservation protocol that could be unavailable at the emitter or receiver side. It must use only the most standard Internet Protocol suite that is IPv4[21].

2.7.2. Solution

A peer modifies an object *X* by multicasting a message [*State*: V_L, G_X, V_X, T_X, S_X]. A receiver uses the event either to create a new object or to update an existing object. The receivers do not acknowledge that message.
When a peer deletes or creates an object, it changes its list (*myList:=myList-X or myList:=myList+X*) and it increases the list version (*myListVersion++;*). Afterwards, the peer multicasts a message [*Deletion*: V'_L, G_X, V_X,] or [*State*: V'_L, G_X, V_X, T_X, S_X] with V'_L=*myListVersion*. A receiving peer removes or adds G_X and updates its local knowledge of the distant list. Thus, emitter multicasts a small deletion or update message instead of using the list messages. The receiving peers do not acknowledge these operations.
When a receiver has versions for the same object ($V'_X \geq V_X$) or list ($V'_L \geq V_L$), it discards the oldest one (V_X or V_L) and it processes uniquely the freshest one (V'_X or V'_L).

An object or list update is transmitted using UDP multicasting. UDP provides a good throughput. Moreover, multicasting consumes less bandwidth and is generally faster than several point-to-point transmissions. At last, emitter does not wait for positive acknowledgements and receiver speeds up recent changes by discarding old updates. Thus, the response time is shortened as much as possible using a standard IPv4 suite.

2.8. Consistent Progression of the Work

2.8.1. Problem

The system must guaranty a consistent progression of the work. For that purpose, a precise semantic of consistency must be satisfied: any operation carried out on a node must start from the latest state of that node. Thus, a designer can observe the current state of the concerned object. So, he decides in context to continue or discard his operation.

2.8.2. Solution

The ownership transfer guaranties that a modification is processed starting from the latest state of the corresponding object. The owner refuses the ownership transfer when a write attribute is disabled. Otherwise, the requesting peer receives at a time the current state and ownership.
A transfer is running in three phases. First, the requester multicasts a message to locate the owner. Second, the owner replies to the requester with a point-to-point message including the ownership and the current state of the node. Third, when receiving the reply, the granted peer sends a point-to-point acknowledgement that terminates the transfer.
The ownership transmission must be reliable. For that purpose, a local number is associated with the request. The owner replies with the same number. When receiving the reply, the requester sends an acknowledgment including the same number. The requester resends his request in absence of response. The granting peer resends the reply in absence of acknowledgment. Faulty situations, where an object is without any owner due to a transmission error, are thus avoided.

Left side of figure 3 presents a solution that does not guaranty a consistent progression of the work. At time *T*, user *1* updates the object *X* with value *B*. At time *T'*, user *2* speculates about the exact state of *X* and he updates *X* with value *C* before observing the update from designer *1*. User *1* lost the work he did on object *X* because of the concurrent update of user 2. He has to redo the work since he is not satisfied with the work of user *2*. One could assume a possibility to recover state *B* on site *2*. But, it is not feasible if state *B* already has been replaced with another state *C*. Moreover, seeing a transient state (*B*), that deletes his work, would be a very unpleasant distortion for user *1*. Putting a lock on the object does not solve the problem (i.e. at time *T*, user 1 has the lock and user 2 acquires the lock at time *T'* but the problem stays).
The right side of figure 3 presents how the ownership transfer solves the problem. At time T', user 2 wants to modify object *X* and recovers first the current state *B* through the ownership transfer. Afterwards, user *2* can decide to continue or to abort

his update. The main difference is that user 2 observes the current state before deciding an update. Thus, he will decide in context and speculation about the current state of *X* is avoided.

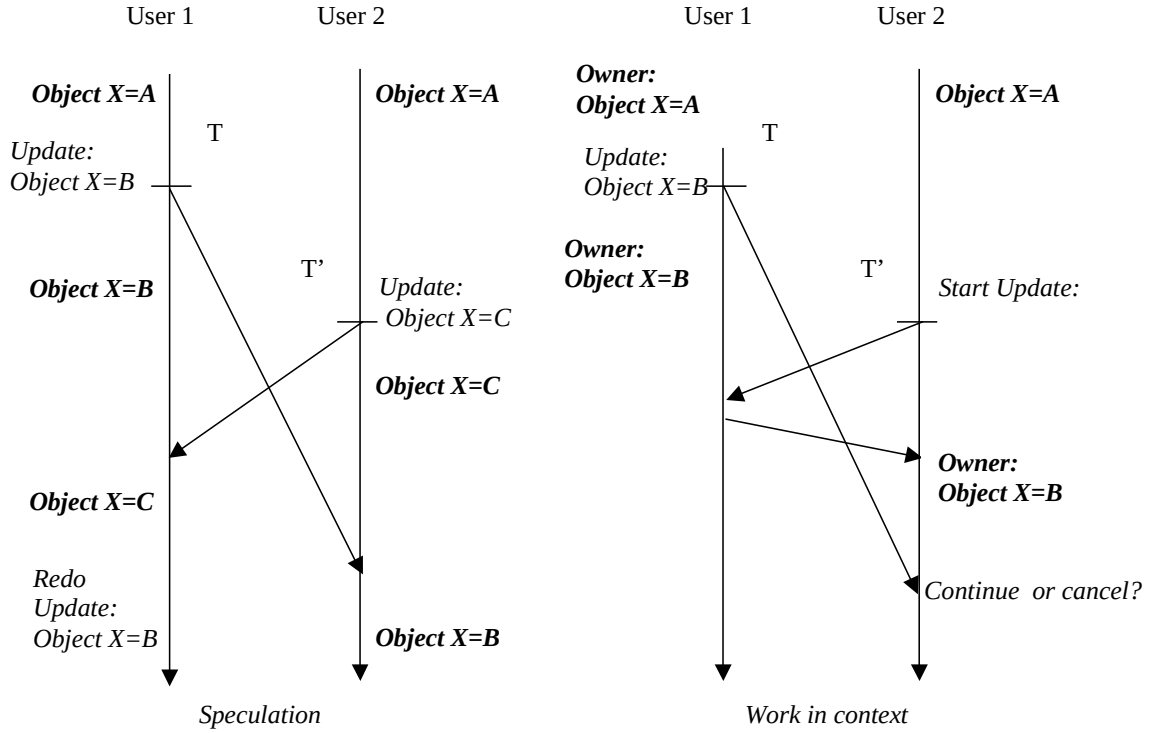

Fig. 3 Consistent progression of the work

It must be noted that a consistent progression is guarantied for a composite operation involving several nodes. In that case, an ownership transfer is processed for each node of the composite operation. Thus, a user gets an exact context for all the concerned nodes before processing his interaction.

2.9. Parallel Work

2.9.1. Problem

Parallel working must be supported both during meeting and during disconnected work. A high level of parallelism must be supported while providing a consistent progression.

2.9.2. Solution

Two tasks dealing with two distinct nodes are processed simultaneously. Our fine grain scene enables at the same time two distant users to carry out operations on two distinct nodes. Any couple of update (i.e. modifying an attribute) is processed simultaneously. Two node creations are processed simultaneously when they do not involve the same father. Two node deletions are processed simultaneously when they concern two disjoint sets of node, father and children.
Two tasks involving the same node are serialized. The current owner processes the first task. Afterwards, the ownership transfer is carried out with the distant task. Thus, the second task starts processing the node after the end of the first task. Serialization is required to guaranty the consistency property. Thus, parallelism is limited only to avoid violation of the consistent progression or distributed designation.

2.10. Refreshment

2.10.1. Problem

A service must enable a peer to resynchronize its copy with fresh state. The solution must consume a small bandwidth.

2.10.2. Solution

To improve the performance the updates and deletions are not acknowledged, a distant peer can thus lose events. Using the refresh service, a peer recovers from missing fresh state in a very efficient way.
To recover fresh states, a peer requests a list to a distant peer. The receiver replies by multicasting the list messages. The requesting peer uses the received list to remove or add objects. If the objects exist already, the received list enables to ask retransmission for newer versions. So, only new missing versions are retransmitted. A retransmission request is a kind of negative acknowledgement (NACQ). But, NACQ are not generated each time a version is missing or to recover all of the states of a given object. NACQ are typically transmitted on user demand or for a periodic checking of the accuracy of distant nodes.
It should be noted that the retransmission request and the reply are multicast. Thus, a peer observing a retransmission request from another peer waits simply the reply. At last, a peer packages different retransmission requests into a single multicast message. Thus, only a small number of retransmission requests are multicast.

Using the list of a distant peer, a peer can request retransmission for new versions that are currently missing. Thus, the requesting peer recovers the latest version from the node owner. This service transmits a small amount of data. First, missing versions uniquely are retransmitted using a negative acknowledgement. Second, uniquely the latest state is retransmitted. Thus, out of date versions are not retransmitted. Moreover, the solution does not ask the emitter to keep the different versions in memory.

2.11. Security

2.11.1. Problem

Confidentiality must be guarantied during the meeting. The solution must provide a solid encryption that makes an attack hard. The solution must not require a peer to be capable of processing X509 certificates.

2.11.2. Solution

Using the session key PK_S, each peer achieves a symmetric encryption of each message. Only authenticated users received the session key securely through S/MIME. So, only authenticated users can decrypt the received messages.
To limit the attacks, a re-keying protocol is developed. The peer with the smallest network address will transmit frequently a new session key PK'_S. The new session key is encrypted using PK_S. The requesting peer multicasts the encrypted message including the new key. If a peer receives messages that cannot be decrypted, it means that the peer misses the new key. In that case, the missed peer asks a retransmission using the previous key PK_S. Each entity maintains the previous key until it observes a message encrypted with the new key from each distant peer.

Confidentiality is provided only with a symmetric encryption. The re-keying protocol does not use X509 certificates to protect the distribution of the key. The system limits the ability to guess the session key since the key changes frequently. To make an attack, a hacker must discover the key quickly. For stronger protection, a Deffie-Hellman key exchange[11.] can be used to share a secret between each pair of participants. The shared secret is then used to exchange the new key PK'_S through point-to-point transmission.

2.12. Address Allocation

2.12.1. Problem

Since there are no standardized solutions to reserve a multicast address, a solution must be provided to allocate dynamically a free address during a meeting. Moreover, allocated address must be securely exchanged between the participants.

2.12.2. Solution

If another application uses the same multicast address, the system will detect automatically the situation because the message cannot be decrypted using the session key PK_S. The peer with the smallest network address looks for a free address by listening during a period if there is no activity on that address. Afterwards, the peer transmits the new address to the

distant peers. Receiving the proposed address, each peer will join the new multicast address. Each peer will listen the two addresses until it receives a message from each peer on the new address. Afterwards, the peer forgets the previous address. Since the proposed address is encrypted, this mechanism enables to allocate and to exchange securely a new address during the meeting.

2.13. Multicast Connectivity

2.13.1. Problem

The IP network does not guaranty a correct routing of the multicast addresses. So, there is no guaranty that a message reaches all the participants. The system must solve the problem and switch automatically in a point-to-point transmission. Moreover, the solution must deal with stations assigning dynamically their network address.

2.13.2. Solution

The system detects automatically if a distant participant is not accessible with multicast. When a peer *A* reachs a meeting, it sends an email to the group of participants including its current *IP* address. In the same time, *B* waits for multicast messages to decide if *A* can be reached through multicast.
If multicast is available between *A* and *B*, the session starts without waiting any email.
Otherwise, *B* will receive an email from *A*. *A* and *B* will then communicate through point-to-point transmissions using the discovered IP address. Since mobile workers can receive emails, they will have the possibility to discover the distant IP address that is currently in use by a distant worker. Thus, the solution enables to discover an IP address that is dynamically allocated by a distant participant.

3. CONCLUSION

Our solution is fully distributed. It does not use any server and it requires only best-effort transmission (UDP) available at any Internet workstation. It does not require any synchronized clocks or specific quality of service from the under-laying network. It proposes a distributed protocol to build the meeting membership. The system aggregates automatically different pieces into a global shared scene through a distributed designation mechanism. Thus, each participant has a replica of the shared scene. A refresh service enables to resynchronize the replicas with fresh states. The solution provides a consistent progression of the work. The ownership transfer guaranties that a modification starts from a consistent state. Thus, design in context is supported. A point-to-point protocol provides a reliable ownership transfer with a low overhead. Parallel working is fully supported since distinct nodes of the scene tree are processed simultaneously. Persistency is achieved in a distributed way within the different local stores. Security is integrated through a symmetric encryption using a session key that is transmitted securely using standard email service. Thus, any mobile worker can receive the shared secret of a future meeting. During a meeting, the system allocates a free multicast address. Thus, mobile workers do not have to reserve the multicast address within a widespread network. Multicast or point-to-point transmission is automatically selected according to the connectivity observed at the mobile worker. Thus, a mobile worker does not have to worry about the multicast connectivity available at the current access point. At last, meeting and disconnected work are supported. Thus, a mobile worker mixes disconnected and collaborative works. Since a small amount of data is transmitted, mobile designers can use the system on a wide-area network with slow communication lines. The distribution services are proposed as a C++ library that any standalone application can use. It runs on Windows and Unix. Our collaboration services are integrated within a virtual prototyping application that is used at EADS within a geographically widespread and mobile team.

REFERENCES

1. Abdel-Wahab H., Favereau J., Kim O., Kabore P., *An Internet Collaborative Environment for Sharing Java Applications*, IEEE Computer Society Workshop on Future Trends of Distributed Systems (FTDCS'97), Tunis, 1997.
2. Barrus J.W., Waters C., Anderson D.B., *Locales: Supporting Large Multi-user Virtual Environments*, IEEE Computer Graphics and Application. November, Vol.16, No.6, pp.50-57, 1996.
3. Brutzman D., Zyda M., Watsen K., *virtual reality transfer protocol (vrtp) Design Rationale*, Workshops on Enabling Technology: Infrastructure for Collaborative Enterprises (WET ICE): Sharing a Distributed Virtual Reality, Cambridge Massachussets, June 1997.

4. Broll W., *DWTP-An Internet Protocol For Shared Virtual World*, International Symposium on the Virtual Reality Modeling Language VRML'98, ACM SIGGRAPH, conference proceedings, pp.49-56, 1998.
5. Costantini F., Sgambato A., Toinard C., Chevassus N., Gaillard F., *An Internet Based Architecture Satisfying the Distributed Building Site Metaphor*. IRMA2000 Multimedia Computing Track, Anchorage, Alaska, 21-24 May, conference proceedings, pp.151-155, published by IDEA Group Publishing ISBN 1-878289-84-5, 2000.
6. Dassault Systems CATIA Marketing Team, *4D Navigator*, http://www.catia.ibm.com/prodinfo/, 1998.
7. Deering S., Hinden R., *Internet Protocol Version 6 (IPv6) Specification*, Request for Comments 1883, December 1995.
8. Dusse S., Hoffman P., Ramsdell B., Lundblade L., Repka L., *S/MIME Version 2 Message Specification*, Request for Comments 2311, March 1998.
9. Defense Modeling and Simulation Office, *HLA Data Distribution Management Design Document Version 0.5*, February, U.S. Department of Defense, Washington D.C., http://www.dmso.mil/project/hla, 1997.
10. Defense Modeling and Simulation Office, *HLA Time Management Design Document V1.0, August*, U.S. Department of Defense, Washington D.C., 1996 .
11. Diffie W., Hellman M., New Directions in Cryptography, IEEE Transactions on Information Theory, November 1976.
12. IBM/Dassault Systems, *ENOVIAPM Version 3 Release 4*, http://www.ibmlink.ibm.com/usalets&parms=H_200-187, June 2000.
13. IBM/Dassault Systems, *ENOVIA Portal Solutions Version 5 Release 4*, http://www.ibmlink.ibm.com/usalets&parms=H_200-191, June 2000.
14. IBM/Dassault Systems, *ENOVIAVPM Version 1 Release 3*, http://www.ibmlink.ibm.com/usalets&parms=H_200-191, June 2000.
15. Greenberg S., Roseman M., *Using a Room Metaphor to Ease Transitions in Groupware*. Research report 98/611/02, University of Calgary, 1998.
16. Hagsand O., *Interactive Multiusers VEs in the DIVE system*, IEEE Multimedia, Vol.3, No.1, pp.30-39, 1996.
17. Handley M., Crowcroft J., *Network Text Editor (NTE) A scalable shared text editor for the MBone*. Proceedings of ACM Sigcomm 97, Canne, France, 1997.
18. Hamilton G., *The JavaBeans API specification*, Sun Microsystems, 1997.
19. Hewlett-Packard, *CoCreate OneSpace: the Award-winning, Web enabled, CAD Independent, real-time Collaboration Solution*, 1999.
20. IEEE Std 1278.1, *Standard for Distributed Interactive Simulation Applications Protocols*, IEEE Computer Society, 1995.
21. Postel J., *Internet Protocol*, Request for Comments 791, September 1981.
22. Kent S., Atkinson R., *Security Architecture for the Internet Protocol*, Request for Comments 2401, November 1998.
23. ITU-T, *Recommendation X.509 (1997 E): Information Technology - Open Systems Interconnection – The Directory: Authentication Framework*, June 1997.
24. ITU-T, *Recommandation H.323 – Packet-based multimedia communications systems*, September 1999.
25. ITU-T, *Recommandation H.235 – Security and Encryption for (H.323 and other H.245-based) multimedia terminals*, February 1998.
26. Sun Microsystems, *Java Shared Data Toolkit*, http://www.sun.com/software/jsdt/index.html, 1999.
27. D. Jefferson, "*Virtual Time*", ACM Trans. on Programming Languages and systems, Vol.7, N°.3, pp404-425, 1985.
28. Kuhmünch C., Fuhrmann T., Schäppe G., *Java Teachware - The Java Remote Control Tool and its Applications*. In proceedings of ED-MEDIA'98, 1998.
29. Lamport L., *Time, clocks and the ordering of events in a distributed system*. CACM Vol 21, Number 7, July 1978.
30. Leigh J., Johnson A.E., Defanti T.A., *Issues in the Design of a Flexible Distributed Architecture for Supporting Persistence and Interoperability in Collaborative Virtual Environments*, Supercomputing, conference proceedings, pp.15-21, 1997.
31. Object Management Group, *Control and Management of A/V streams specification*, OMG Document telecom/97-05-07 edition, 1997.
32. Object Management Group, *The Common Object Request Broker: Architecture and Specification*, Edition 2.3, 1999.
33. Tie Liao, *Light-weight Reliable Multicast Protocol*, http://webcanal.inria.fr/lrmp/index.html, 1999.
34. Macedonia M.R., Zyda M.J., Pratt D.R., Brutzman D.P., Barham P.T. 1995. *Exploiting Reality with Multicast Groups*, IEEE Computer Graphics and Applications, Vol.15, No.5, pp.38-45, 1995.
35. Microsoft, NetMeeting, http://www.microsoft.com/windows/NetMeeting/Features/default.ASP, June 1999.
36. Misra J., "*Distributed Discrete-Event Simulation*", ACM Computing Surveys, Vol. 18, N°. 1, pp.39-65, 1986.
37. Eleftheriadis, A., Herpel, C., Rajan, G., Ward, L., *Text for ISO/IEC FCD 14496-1 Systems*, ISO/IEC JTC1/SC29/WG11CODING OF MOVING PICTURES AND AUDIO, May 1998.

38. Postel J., User Datagram Protocol, Request For Comments 768, 28 August 1980.
39. Postel J., *Transmission Control Protocol*, Request For Comments 793, September 1981.
40. Sun Microsystems, *Java Remote Method Invocation (RMI)*, http://www.javasoft.com/j2se/1.3/docs/guide/rmi/index.html, 1999.
41. Roseman M., Greenberg S., *Building real time groupware with GroupKit, a groupware toolkit*, ACM Transactions on Computer Human Interaction, 3(1), pp. 66-106, 1996.
42. Braden R., Zhang L., Berson S., Herzog S., Jamin S., *Resource ReSerVation Protocol (RSVP) Version 1 Functional Specification*, Request for Comments 2205, September 1997.
43. Saddik A., Karaduman O., Fischer S., Steinmetz R., *Collaborative Working with Stand-Alone Applets*. Proc. Of Intelligent Mutimedia and Distance Education ISIMADE'99, Baden-Baden, Germany, 1999.
44. Handley M., Jacobson V., *SDP: Session Description Protocol*, Request For Comments 2327. 1998.
45. Schulzrinne H., Casner S., Frederick R., Jacobson V., *RTP: A Transport Protocol for Real-Time Applications*, Request For Comments 1889, January 1996.
46. Shirmohammadi S., Oliveira J.C., Georganas N.D., *Java-Based Multimedia Collaboration: Approaches and Issues*. Proc. International Conference On Telecommunications (ICT '98), Vol. I, Porto Carras, Greece, 1998.
47. TeamWave, *TeamWave Workspace Overview*, http://www.teamwave.com/advantages/index.html, April 2000.
48. Dierks T., Allen C., *The TLS Protocol Version 1.0*, Request for Comments 2246, January 1999.
49. Toinard C., Chevassus N., *Virtual world objects for real-time cooperative design of manufacturing systems*, Lecture Notes in Computer Sciences, Object Oriented Technology ECOOP'98 Workshop Reader, Springer Verlag, conference proceedings, pp. 525-528, 1998.
50. Toinard C., Florin G., Carrez C., *A Formal Method to Prove Ordering Properties of Multicast Systems*, ACM Operating Systems Review, Vol. 33, No.4, pp.75-89, 1999.
51. Toinard C., Chevassus N., Sgambato A., *Collaborative Virtual Reality and the Distributed Building Site Metaphor*, WSCG'99, The 7-th International Conference in Central Europe on Computer Graphics, Visualization and Interactive Digital Media'99, conference proceedings, pp. SP/111-117, 1999.
52. Wahl, M., Howes, T., Kille, S., *Lightweight Directory Access Protocol (v3)* Request for Comments 2251, 1997.
53. Sense 8, *World2World Release 1 Technical Overview*, 1997.
54. W3C, *Extensible Markup Language (XML) 1.0*, W3C Recommendation REC-xml-19980210, 10 February 1998.