# COLLABORATIVE VIRTUAL REALITY AND THE DISTRIBUTED BUILDING SITE METAPHOR

**Christian Toinard[#], Nicolas Chevassus[+], Antoine Sgambato[#]**

[#]CEDRIC – Centre D'Etudes et de Recherche en Informatique CNAM
292 rue Saint-Martin 75141 Paris Cedex 03 France
toinard@cnam.fr, sgamba_a@cnam.fr

[+]AEROSPATIALE – Centre Commun de Recherches
12 rue Pasteur BP 76 92152 Suresnes Cedex France
NICOLAS.CHEVASSUS@siege.aerospatiale.fr

**ABSTRACT**

Virtual reality is used to support the collaboration in design of manufacturing products. Most of the time, solutions focus on sophisticated communication architectures in order to satisfy general requirements of collaboration. Within the building site, workers share and split dynamically the work, join their local works automatically into a global work and achieve conciliation. The corresponding services are provided through low cost communication services available on standard Internet workstations. A generic solution allows distributing dynamically the scene graph beyond the different users. The users form a meeting to interact on the global scene while keeping the work consistent. They can resolve the choices of design through a conciliation procedure. The solution allows a low cost deployment and highly usable collaboration services. In contrast with other solutions, the collaboration services fit the design activity of manufacturing products and only use best effort communication services.

**Keywords:** distributed virtual reality, collaborative design, fast prototyping, collaboration services, communication services.

## INTRODUCTION

Currently, distributed virtual environments address poorly the collaborative design activity. The paper presents the distributed building site metaphor and how to implement that metaphor. Historically, distributed virtual reality is associated with simulation of battlefields like [Maced95]. These systems reduce the network traffic due to a high number of moving objects. Generally, a way to divide statically the scene is provided either for scalability or persistence. [Barru96] proposes to divide a large virtual world into different locales according to a geographic cutting. The users can combine their separate locales together. The combination corresponds to a connection of the different locales (each locale can be seen like a room). A user is aware of the modifications carried out on neighboring locales. A user can move to any neighboring locale. The work is consistent because a locale belongs to a single user (i.e. the creator maintains the locale). Thus, the locales are distributed statically beyond the different users. Any user can place a new object into a locale. He is responsible for the maintenance of his object. So, an object statically belongs to his creator. Like [IBMDa98], industrial solutions provide a collaborative navigation of the scene. Thus, a simple form of design review is achieved. GroupWare toolkits [Rosem96] [Netsc98] do not manage a virtual scene. Moreover, [Netsc98] provides audio transmission and integrates the application [IBMDa98]. That way, scene navigation starts from a group session.

## DISTRIBUTED BUILDING SITE REQUIREMENTS

The distributed building site metaphor is the model that we want to realize. The building site is a shared place where the workers meet to reunify a global work. A user brings pieces of work outside the meeting. The requirements provide important leverage on the problem of collaboration.

### Model Requirements

Application tree: the application maintains a scene tree that is composed of different objects. The scene tree is an abstraction of the application scene. That scene describes graphical information but also data that are relevant for the design activity.

Hierarchical object: each object defines a tree where the leaves are elementary nodes (i.e. a beam or an electrical route).

Object protection: An object and elementary nodes is associated with protection attributes (read right and write right). Thus, a user prevents or authorizes the modifications from the other users.

Shared workspace: the scene tree is maintained by the system. It is accessible during a meeting between several users. The different users enter independently into a meeting. The meeting allows the users to reform the global scene tree by unifying their isolated workspaces. The global tree reforms automatically. Each user creates, deletes or modifies nodes in line with the protection rules.

Isolated work space: a user can work locally on a subset of the shared space. A user creates, deletes or modifies nodes without any communication at all.

Isolated nodes: when leaving a meeting, a user defines which nodes he wants in his isolated workspace. At the time of leaving, two kinds of nodes can be distinguished. The nodes owned by the user are brought into his isolated space. The other nodes can be selected by the user in order to get a clone or a duplicate in his isolated space.

Parallel working: during a meeting two nodes owned by two different peers are processed in parallel without requiring any communication. Moreover, two isolated spaces provide a parallel working that does not use any communication at all.

Concurrency control: the system supports concurrent requests for the same node. These concurrent requests are serialized in order to maintain the consistency. Consistency is provided in a distributed way through the ownership transfer.

Shared work consistency: the work remains consistent because any modification is achieved from the last state of the considered node. The owner grants the state to a requesting peer.

Real time awareness: during a meeting, any participant observes the distant interactions as quick as allowed by his processing speed. Event passing authorizes a synchronization of the distant peers. The transmission time is as short as possible. A peer speeds up the delivery of recent updates.

Distribution of the scene tree: the global scene tree is not maintained on a central server. It is completely distributed over the different users through node relationships. During a meeting, the global tree reforms automatically using the local relationships of each owner. The scene can be processed as a whole when all the users are present.

Dynamic distribution of the scene tree: During a meeting, the ownership of any node is distributed dynamically according to the user requests. So, the management of a node is distributed dynamically.

Conciliation: during a meeting, the participants use a conciliation service to combine gracefully their separate works. They define the modification that must be achieved on a given object or node.

Work persistency: different levels of persistence are provided from participatory persistence to continuous persistence.

Full distribution: the solution does not rely on any central server. A server does not manage the shared workspace. Consistency is not provided through a state server or a lock server for the concerned object. Persistence is completely distributed. The system does not use any server to manage the arrivals and departures of the participants.

### Key Benefits

Aggregate and scattering of objects: the participants combine their elementary objects into different aggregates. An aggregate is not a juxtaposition of different rooms of work using transformation matrixes to connect them. An aggregate represents the organization of a complex object through a global hierarchical structure of nodes. An object can be scattered into different isolated workspaces. It belongs to a global hierarchical tree. The aggregate reforms later on automatically.

Advanced collaboration for the design activity: the collaboration is not limited to a collection of different separate works. When a user joins a meeting, he may permit the other participants to

modify his work. The isolated work supports the classical way workers prepare their tasks before reaching a building site (e.g. the different beams of a mechanical structure are assembled before entering into the building site). Moreover, the isolation enhances the collaboration because distinct propositions can be made for the same work (e.g. two workers prepare two propositions for a given structure).

Efficient collaboration: a participant authorizes or prevents other participants to modify the owned nodes by setting properly their protection attributes. The protection attributes can be displayed and set like any other attributes (i.e. size, color, position, etc…) through a scrolling interface. Real time interactions are inline with the protection. Moreover, the isolation allows improving the design as a long term job without disruption due to distant interactions or faults.

Communication efficiency: an application peer processes an application tree and uses the collaboration services. Thus, peers only exchange events that are relevant for the application and collaboration. In contrast, it is obviously inefficient to consider that each elementary event (like a mouse pointer motion or a VRML node update) must be sent on the network. The application tree drastically reduces the number of events that must be process by a peer. Bandwidth reservations and solutions to reduce the communication load become useless.

High performances and consistency: parallel working is due to the distribution of the application tree and to the local nature of the relationships between the nodes. Distinct tasks are processed simultaneously without communication between them. A consistency protocol is only processed when a parallel working can not be supported due to the requirement of consistency. Moreover, only the requesting peer and the granting peer are involved. Most of the time, the collaboration systems limit the ability of parallel working or do not guaranty the consistency. Our metaphor supports parallel working and a state transfer is sufficient to recover the inconsistencies when required.

Fast animation of the scene: in order to achieve a real time awareness, events are transmitted to the distant peers. Since the transmission time is shortened as much as possible and the application uniquely processes recent states, the distant participants observe a fast animation of the shared space. Event passing does not address the parallel working but uniquely the real time awareness.

Freedom of conciliation: conciliation is as loose as possible. Thus, the participants can freely decide which operation must be processed for the shared work. A more constrained way of making conciliation would not have a sufficient usability. The general nature of the conciliation procedure also makes it possible to recover possible remaining faults (from failstop to byzantine faults).

Fault resistance: since the system does not use any server, the work can go on despite the fault of distant machines. For example, two nodes of the same object can evolve independently, during the same meeting, despite the fault of the other one. Moreover, an isolated workspace is not affected by the faults of the other workspaces.

Efficient persistence: a user only saves the nodes which he is the owner. The persistence is carried out in a distributed way by the different local savings.

## RELATED WORKS

### Locales

[Barru96] allows combining different rooms into a global world. The combination relies on transformation matrixes in order to connect adjacent rooms that present communication doors between them. A room is called a locale. Thus, a user located in a room can observe the events of an adjacent room through the door. The transformation matrix allows representing the neighboring rooms and moving smoothly from one room to another. In that context, a neighboring room can be connected dynamically in order to combine the works achieved separately in different rooms. The participants use different locales to work separately. This solution provides three interesting features: graceful motion between neighboring locales, performances are improved through decomposition of the world into chunks that can be processed separately and workers combine separate objects through a dynamic connection of their locales. When users build separate elementary objects, these objects are not aggregated through a global scene tree. A locale is assigned to a server maintaining its state.

### High Level Architecture

[DMSO97] addresses the framework of a great number of moving objects. The solution aims at a real time that is as much as possible exact. They provide sophisticated solutions to reduce the number of state transmissions and to recover the transmission errors. [DMSO97] defines a dynamic cutting (which is not perfect) according to the motion of the objects. Each moving object defines an area of interest and events falling into that area

are observed by the moving object. The difficulty is to change dynamically the area of interest according to the motion of the object. A given object does not have the guaranty to receive all the interesting events. Moreover, this standard defines different ordering of events that can be used in a simulation context. [DMSO96] identifies different qualities of ordering. They are mainly interested with a logical ordering of events where the application associates logical dates to the events. That ordering supports a distributed simulation by guaranteeing that the events are processed at the corresponding logical time. These properties do not address a real time simulation or a collaborative activity. Ownership services are defined. The ownership management allows federates to transfer ownership of object attributes. The ownership transfer is associated with publication and subscription services. Ownership acquisition requires current publication and subscription for the attribute. The standard does not define different levels of ownership like read and write permissions. It is not integrated in fine grain collaboration where a complex hierarchy is defined for each object. The ownership management concerns less the cooperative design than the distributed simulation. That standard does not standardized the protocols in order to get a freedom of solutions. In practice, a run-time [Calvi97] is under development. It assumes resource reservation (i.e. ReserVation Protocol) and reliable multicasting. These requirements seem necessary in the event of a great number of state transmissions associated with moving objects.

**Distributed Interactive Virtual Environment**

[Hagsa96] also addresses moving object and aims at a precise real time. Solutions reduce the number of state transmissions. In that context, uniquely fresh events interest the recipient and the system improves the performances relying on that feature. With DIVE, the application has the responsibility of the placement of the copies for the shared world. It must define which object must be present in the different copies. The consistency of the copies is not addressed. This solution provides a protocol allowing to localize the nearest copy for a given object. When an entity detects a missing event or just requests an object, it multicasts a request. The closest copy replies with its latest version of the object. The latest version of the object is not the freshest one. Another copy can have a better knowledge for the state of that object. When nobody have a better proposition, only the closest copy replies to the request. The system implements a classical dead-reckoning module, like [IEEE95], in order to keep the number of updates low: between two updates, a receiving entity evaluates the position using linear and angular velocity.

**Distributed World Transfer Protocol**

[Broll98] considers a server for: transmission of the virtual world contents, reliability of the updates, persistence, supervising the connections of participants, supporting consistency. In essence, that solution is close to [Hagsa96]. Proxies manage copies of the shared world to reduce the load of the server. The server and the copies are updated when a peer multicasts events. The system provides a recovery mechanism. Each peer has the ability to receive all the events. When the recovery can not be achieved, a participant has to reconnect to the server. [Broll98] provides a simple concurrency control achieved through a lock mechanism. The approach is a locking on a per-interaction basis instead of a per-object basis. The server releases locks within a certain time. That concurrency control suits well for special requirements (like controlling concurrent motions of the objects). It does not guaranty consistency. [Broll98] integrates solutions to limit the bandwidth that is necessary when working at a VRML level of abstraction. The system improves the performances using cells. The cells define statically a mapping of the world. Each peer has to connect to a central server. The server transmits mapping and world state at the connection time. The solution does not describe how to add dynamically new cells to the world.

**Cavern**

[Leigh97] addresses design review and cooperation services with immersion in the virtual world. A client uses a server with supercomputing resources and high-performance networking. This centralized solution requires resource reservation and reliable transmission. Through a persistent object server, the system supports state persistence. Either intermittent snapshots can be created or entire collaboration experiences can be recorded for later review. A client must specify a Quality of Service (QoS) in order to have the desired bandwidth and jitter. These requirements are due to the nature of cooperation where immersion generates a great volume of data with real-time constraints. The solution aims at loading large scientific data (like video or CAD format). Caches using timestamps improve the performances. They do not tackle the collaboration between users where a global scene can be split and reunified during a meeting.

## CONTRIBUTION OF THE IMPLEMENTATION

Before describing the solution, let us resume the contributions. The system only uses best effort

communication available on any standard IP (Internet Protocol) machine with UDP (Unreliable Delivery Protocol) and multicast ability. For the ownership transfer, a protocol simply recovers the transmission errors through retransmission using a local date. The services do not implement a reliable multicasting for the event transmissions. Systems building a reliable multicasting are confronted with the ACK implosion problem that is still an on going work. Here, the errors do not affect the work consistency because the owner always has a consistent state for the considered node. The inconsistencies are recovered when required through the ownership protocol. Thus, the requester becomes consistent before processing the node. This is a low cost solution where the inconsistencies are recovered uniquely when necessary. Moreover, there is a well-known impossibility to recover all the faults and to maintain the participants consistent. Ordering properties like causal and total ordering [Flori92] introduce higher costs than a reliable multicast. Ordered multicasts are well known to scale poorly. Moreover, they only capture the causal relationships from the application that are associated with communication causal relationships [Lampo78]. Ordered multicasts introduce high overhead and do not guaranty to preserve the interesting relationships. Generally, fault-tolerant applications try to use them. Our system treats the causal relationships at low cost with application messages using the conciliation procedure. The basic idea is to circumvent these ordering problems with a reliable and low cost consistency protocol used at the right time.

## PRINCIPLES

Each application peer uses collaboration services to maintain a copy of the scene. Let us describe how these services satisfy our model.

### Distributed Application Tree

The shared work composes an application tree that can express a refinement purpose, a structural purpose or any organizational purpose. Figure 1 shows a simplified application tree dealing with an aircraft subassembly design. Starting form a state where user A owns the whole equipped structure, B the whole air system and C the whole hydraulic system, the tree is distributed as follows at a given time during a meeting: user A owns the nodes {1, 1.1, 1.1.1, 1.1.3, 3.1, 4.1.2}, user B owns the nodes {1.2, 1.1.2, 1.2.1, 3, 3.3}, user C owns the nodes {3.5, 4, 4.1}. Simultaneously, A can update the floor 1.1.1 while B updates the beam 1.1.2 of the same structure. They can work on that global tree on an isolation or meeting basis.

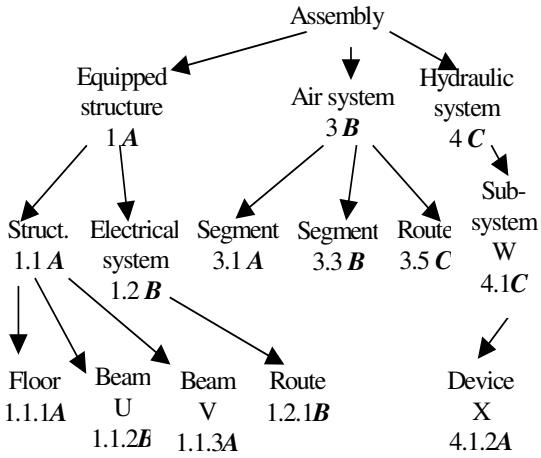


Figure 1

### Isolation

Whenever a user works outside a meeting time, he can improve the global tree without interactions from others. First, the isolated worker does not observe the operations carried out on the remote nodes. Second, the distant workers do not observe any of his operations. Aiming at joining a meeting, the isolated worker associates appropriate protection attributes to each node. He can deprive other workers from observing particular nodes or authorize them to observe or modify them later on. The protection, defined during isolation, uniquely takes effect during a meeting. For example, user C works isolated on the nodes shown in figure 1. He can modify the design of route 3.5. He can add devices to the subsystem 4.1 or create a new subsystem in the hydraulic system 4. These features do not use any network communication (the peer runs like a local application).

### Meeting

When joining a meeting, a participant discovers the name of other participants. He observes the global tree as a whole when all the participants are in the meeting. Otherwise, he simply observes a subset of the global tree. A participant has the ability to hide nodes from others by disabling their read and write attributes before entering into the meeting. During the meeting, the owner can change the protection rules. Thus, hidden nodes can be discovered, discovered nodes can be hidden, and distant modifications can be enabled or cancelled.

Let us describe briefly the dynamic distribution. Whenever the write permission is enabled, a participant can modify a node that does not belong to him. The associated peer requests the ownership.

The peer of the current owner grants the ownership to the requesting peer. Thus, the right of modifying a node is distributed dynamically according to the user interactions. Each peer has a copy of the world. That copy contains a state for nodes owned by the peer and nodes owned by distant participants. The dynamic distribution does not change the number of nodes inside a peer. It changes simply the set of nodes owned by the requester and granting peer. For the ownership transfer, a request is multicast to a class D address through UDP (see RFC 1112 [Deeri89] for further details). The reply uses a point to point transmission through UDP. Reply contains the current state of the node within generally few bytes. Whenever a participant leaves a meeting, the set of the owned nodes will serve as a set for his isolated work. In order to bring at home a good view of the tree, the leaving participant has the ability to require a clone for other nodes. A clone is build locally. It remembers the original node. A clone can not be modified. If a participant wishes to bring an unauthorized node in order to modify it, he duplicates the node. The duplicated node remembers the original node. In contrast with a clone, the owner can modify a duplicate.

For example, A and B decide to form a meeting. Before entering into the meeting, user A disables the read attribute of beam 1.1.3 because he did not finished the design. The two participants observe the same nodes except hidden nodes. Thus, B does not observe beam 1.1.3. Nodes from C can not be observed. Despite the absence of C owning the node 4.1, the two participants observe the device 4.1.2. Thus, A and B can modify device 4.1.2. A and B perform real-time modifications of floor 1.1.1, beam 1.1.2 and route 1.2.1. The two participants observe these improvements. Simultaneously, A changes the design of the floor 1.1.1 while B modifies the beam 1.1.2. The two participants observe these modifications. User A points out beam 1.1.2 whose write attribute is enabled. First, current state of 1.1.2 is transferred to the peer A in case of an inconsistent local state. Second, peer A is the owner and can modify the node. User B observes the modification from A. User B in turn decides to modify the node and gets back the ownership and state of node 1.1.2. Then, A decides to leave the meeting in order to improve locally the floor 1.1.1. He points out beam 1.1.2 in order to get a clone. Thus, his isolated workspace provides a good perception of structure 1.1 as a whole. His isolated work contains the improvements from the last meeting. Afterwards, A and B re-form a meeting to join their mutual work on structure 1.1. The clone is not included in the global work. The new state of beam 1.1.2 is observed and overwrites the clone state.

**Consistency**

The distributed work remains consistent. Consistency has different meanings according to the authors and the application requirements. Here, an interaction always occurs from the last state of the considered node. This implies that a peer receives state and ownership at a time. That way, the application tree (distributed over the subsets owned by the different workers) is consistent. For a given participant, a node that is owned by another participant can be inconsistent due to a transmission error. This inconsistency is not awkward with respect to the design activity. The peer simply has a wrong state for a distant work. The important point is that the node be consistent within the owner's peer. The ownership transfer provides a consistency protocol.

For example, user A is in a state X that is an instantaneous snapshot of peer A. The state of each node has a distinct value $X_1$, $X_{1.1}$, $X_{1.1.1}$, $X_{1.1.3}$, $X_{3.1}$, $X_{4.1.2}$. The states of these nodes evolve independently. The notation is simplified by denoting simply X the state of A at a given time. On the same way, let us assume that user B is in a state Z. Every state maintains by the owner is consistent (i.e. state $X_{1.1.1}$ is consistent in peer A and state $Z_{1.1.2}$ is consistent in peer B). Peer A misses some updates from peer B: node 1.1.2 is still in state $X_{1.1.2}$ while the current state is $Z_{1.1.2}$. Thus, the node 1.1.2 is inconsistent in the peer A. This inconsistency is transient because it will be recovered by the next update. Moreover, it is not awkward because the owner B has a consistent state. If user A points out 1.1.2, it will recover a consistent state before being authorized to modify that node.

**Persistence**

Persistence is provided in a completely distributed way by using local storage facilities. First, participatory persistence is supported. The cooperative work is lost when all participants leave. When the cooperative work is restarted, the preceding state is extinguished. Second, state persistence is achieved with local savings. Each participant saves his work at intermediate points or before leaving the meeting.. At a later time, the cooperation can restart at any given point. Third, continuous persistence uses savings during the isolated work. On an isolation basis, a user works from a preceding point. Afterwards, he has the responsibility to save the work achieved during the isolation time. When reentering into a meeting, his starting state contains the result of previous meetings enhanced with the isolated work.

**Real time awareness**

During a meeting, a user is aware of any distant event within a short time (real time awareness). An event is whatever operation on the application tree and meeting (node creation, node destruction, node update, node notification, participant arrival, participant departure and so on…). Events are transmitted in real time to the peers. They are multicast trough a best effort transmission using UDP. The solution does not require any reliable multicast. Thus, the transmission time is as short as possible. The system does not acknowledge any multicast. But, each event is piggybacked with a local date. The local date can be used to deliver only the fresh events. For this purpose, the application uses a specific service. The system discards the events that the peer does not have time to process before the arrival of the new one. Thus, a peer simply treats the interesting events according to its processing speed. For example, an update that the application did not have time to process is discarded when a new update arrives for that node. Some events that are not discarded are transmitted through standard services (i.e. creation events, destruction events, notification events or participatory events).

**Conciliation**

The procedure must provide a high level of freedom in the way a consensus is reached beyond the participants. It would be very difficult to achieve an automated conciliation that satisfies all the situations. Moreover, the conciliation can also be used to recover from remaining faults. Even if the system does its best to preserve the consistency, provide an accurate awareness and recover the faults, some situations still exist where the system is faulty. This assertion comes from the impossibility of a distributed consensus with one faulty process. So, it is interesting that the conciliation provides a way to define a new state from a disrupted state. Whenever a participant points out a node for conciliation, he provides an annotation for the other participants. The conciliation service multicasts the notification to the peers. A receiving participant observes the annotation and uses the same service to reply.

Let us consider two short examples. First, user A is currently the owner of beam 1.1.2. After improving the beam, user A notifies the electrical route 1.2.1 and posts the following annotation*: Route must be modified*. This annotation can be addressed either for improving the design or for recovering an error. In the first case, B modifies the route 1.2.1. The latter case corresponds to a communication error (as the system does not retransmit events, A can miss route 1.2.1's updates). The owner B can reply: *[Re: Route must be modified ]-> it has been updated according to your beam (if required, use the synchronization service to recover the update)*. This example shows briefly that an application message can capture the causal relationships (the close *[Re:M1]-> M2* defines a causal relation between the two messages *M1* and *M2*). Second, a failstop fault of a peer occurs. In the absence of response, a participant (that interacts with a faulty node) can use the conciliation service. He notifies the faulty node and asks the remaining participants if they want to become owner of the orphan. According to the replies, the participants decide who must become owner.

**Distributed Designation**

The system provides a distributed designation of nodes. At first level, a unique name is attributed locally when a user creates a new node. The unique name contains the address of the creation machine plus a local date (e.g. name *@IPA,dateA* defines the *Equipped Structure 1*). The subsequent nodes are designated according to their position (e.g. *@IPA,dateA,1.4* corresponds to the *Beam 1.1.3* because it is the first branch and the fourth leave that has been created by *A*). Only the owner can create a child node (e.g. A creates node *@IPA,dateA,1.4* from *@IPA,dateA,1 = Structure 1.1*). That way, unique names are defined in a distributed way by the different private spaces (e.g. name *@IPA,dateA,1.4* assumes that A creates *@IPA,dateA,1.4* before deleting a node *@IPA,dateA,1.3*). Thus, names are managed by the owner of the father that has a consistent view of his sons (e.g. A or a new owner of *@IPA,dateA,1* reuses the name *@IPA,dateA,1.3* for a new *Beam W 1.1.4*).

**Retransmission**

In order to achieve correctly the ownership transfer, the system retransmits the requests. Each request is associated with a local date and the reply contains the date of the associated request. This mechanism has a low overhead. The request is multicast and a single peer replies. The reply is used as an acknowledgment. The granting peer remains the identity of the new owner in order to retransmit the reply.

**Synchronization**

A participant uses a start service to enter into the meeting. The service multicasts an arrival event that contains the current state of the new participant (nodes from his workspace). Each peer in the meeting replies with its current state. Thus, a new

participant receives a current state from others. This mechanism avoids a central server to initiate the meeting (generally, the server establishes a connection with the participants, disseminates the arrivals or departures of participants and provides the current states for the distant participants). During the meeting, a participant uses the synchronization service to get a fresh state for a node or participant. The service multicasts a request. The peer, that manages the node or participant, replies with a multicast. The reply contains the state for the node or participant. Thus, the requester can recover a possible inconsistency. At the same time, the other participants can use the reply to synchronize their local state with the received value.

## CONCLUSION

The paper presents the distributed building site metaphor. Collaboration services are defined in order to achieve advanced interactions within a virtual environment. The services provide a dynamic distribution of a global application tree and manage a fine grain collaboration. The world is distributed in different workspaces. These workspaces define a global consistent tree. They can be used on an isolation basis. The global scene reforms automatically within a meeting. Participants control easily the shared work using protection attributes. The collaboration supports parallel working and preserves the consistency. Real time awareness and persistence services are available. Participants use conciliation services to reach a design agreement. The paper also describes the implementation of our model and services. The collaboration system uses a best effort transmission with multicast communications and point to point communications. Instead of a reliable multicasting, an efficient and distributed consistency protocol recovers the inconsistency when required. The solution is completely distributed. It does not present any server having a better knowledge than a common peer. Currently, the services have been integrated within an existing application. That collaborative application is deployed at Aerospatiale for day to day conciliation.

## REFERENCES


Barrus,J.W., Waters,C., Anderson,D.B. 1996 *Locale: Supporting Large Multiuser Virtual Environments*, IEEE Computer Graphics and Application. November, Vol.16, No.6, pp.50-57.

Broll,W. 1998 *DWTP-An Internet Protocol For Shared Virtual World*, International Symposium on the Virtual Reality Modeling Language VRML'98, ACM SIGGRAPH, confer. proceedings, pp.49-56.

Calvin, J.O., Chiang, C.J., McGarry, S.M., Rak, S.J. Van Hook, D.J., Salisbury, M. 1997 Design, Implementation, and Performance of the STOW RTI Prototype (RTI-s), Simulat. Interop. Workshop, http://dss.ll.mit.edu/dss.web/097-019.html.

Deering, S.E. 1989 *Host Extensions for IP Multicasting*, Request For Comments 1112.

Defense Modeling and Simulation Office. 1997 *HLA Data Distribution Management Design Document Version 0.5*, February, U.S. Department of Defense, Washington D.C., http://www.dmso.mil/project/hla.

Defense Modeling and Simulation Office. 1996 *HLA Time Management Design Document V1.0, August,* U.S. Department of Defense, Washington D.C., http://www.dmso.mil/project/hla.

Florin, G., Toinard, C. 1992. *A new way to design causally and totally ordered multicast protocols*. ACM Operating Systems Review, Vol 26, Number 4. pp. 77-83.

Hagsand,O. 1996 *Interactive Multiusers VEs in the DIVE system*, IEEE Multimedia, Vol.3, No.1, pp.30-39.

IBM/Dassault Systèmes CATIA Marketing Team 1998 *4D Navigator*, http://www.catia.ibm.com/prodinfo/

IEEE Std 1278.1. 1995 *Standard for Distributed Interactive Simulation Applications Protocols*, IEEE Computer Society.

Lamport, L. 1978 *Time, Clocks and the ordering of events in a distributed system*. Comm. ACM 21, July, 558-565.

Leigh,J., Johnson,A.E., Defanti,T.A. 1997 *Issues in the Design of a Flexible Distributed Architecture for Supporting Persistence and Interoperability in Collaborative Virtual Environments,* Supercomputing, conference proceedings, pp.15-21.

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.

Netscape. 1998 *Nestcape Conference*. http://home.netscape.com/communicator/conference/v4.0/index.html.

Roseman, M. and Greenberg, S. 1996. Building real time groupware with GroupKit, a groupware toolkit. ACM Transactions on Computer Human Interaction, , ACM Press, 3(1), p66-106.