

# Mu3D: A Causal Consistency Protocol for a Collaborative VRML Editor

Ricardo Galli

Yuhua Luo

University of Balearic Islands

## Abstract

This paper describes the implementation of the Mu3D application protocol and consistency control mechanisms to allow the collaborative editing of CAD design. The collaborative editor (M3D editor) developed by us is VRML compliant. The editor has been used as a base for the European Esprit project No. 26287 - M3D and the Spanish project TEL 96-0544/CODI for Cooperative CAD applications.

In our system, only the changes to local databases are transmitted to other collaborative session members. To assure database consistency, the system provides consistency control over the shared data space. A great effort has been paid also to provide a high capability of cooperation and user interactivity while narrowing networks bandwidth requirements.

**CR Categories and Subject Descriptors:** I.3.2 [Computer Graphics]: Graphic Systems - *Distributed/network graphics*. I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism - *Virtual Reality*. J.6 [Computer Applications]: Computer Aided Engineering - Computer-aided design (CAD). C.2.2 [Computer Communication Networks]: Network Protocols. C.2.4 [Computer Communication Networks]: Distributed Systems - *Distributed applications*.

**Additional Keywords:** distributed virtual environments, multicasting, VRML, architecture, CAD.

---

Mathematics and Computer Science Department.  
University of Balearic Islands.  
Carr. de Valldemossa km 7.5, E-07071, Spain.  
e-mail: gallir@m3d.uib.es / yuhua@m3d.uib.es

## 1 INTRODUCTION

The M3D system is a higher level CSCW system particularly for architectural production [Luo99a, Luo99b]. The main component of the M3D system is the collaborative VRML based editor, the M3D Editor. Architectural designs are converted to VRML format using our own developed or third parties tools [Gall97, Sall97], [Cad1]. Its capability of supporting the cooperative work not only includes the distributed-collaborative visualisation, but also, and more important, the on-line modification of CAD objects for an architectural design team. A shared, VRML compatible<sup>1</sup> database is also developed for information storing and retrieval of the whole architectural project.

The objective of the design and development of the M3D Editor was twofold:

1. Provide a rich set of editing operations focused on architects and engineers necessities.
2. Allow the system to work through low to medium bandwidth networks. Our major target is to achieve good interactive performance in ISDN connections because it is the standard set-up for European architecture small and medium size enterprises.

Although centralised system architecture is easier to implement and to control database consistency, the second objective makes us impossible to use this architecture. The architecture we implement is fully distributed. The applications and data are replicated among all sites participating in a collaborative session.

To send local changes to the other database instances we developed an application communication protocol specially suited for transmitting VRML node changes in small messages: the Mu3D (Multi-user 3D) protocol.

In order to maintain the consistency of the database, we have to assure that all messages are sent and delivered to the peer editors in the right order. Because the editor uses a general platform for collaborative work (JESP) [Alme95] that does not provide total ordering message delivering, only source ordered delivery, we have to implement policies and mechanisms to assure causal ordering delivery (or synchronisation).

A very natural but naive idea would be a mechanism that requires an ACK message for every update message. However, this will lower the interactivity and increase the network traffic.

---

<sup>1</sup> It allows to identify stored VRML files through a "unique" URL.

Instead, we decide to use the causal ordering. This will allow the editor to update the local copy and allow the user to continue interaction immediately. Special “barriers” are needed to assure causal synchronisation. These barriers are short periods of time that occurs when a user changes his/her “interest” to another object. During this short period, we force all the replicas to check their local database and to accept or not the selection (through ACKs messages, similar to a two-phase handshake).

We call this technique the “selection” policy [Luo98] (or better to say “selection-and-go”). In order to allow an object to be modified, the user has to select first the sub-tree defined beneath such object. If the selection is ACKed by every site, the object is “locked” and can be just modified by the site maintaining the locking. The object is released when the user de-selects the whole sub-tree.

In the next sections, we shortly describe the architecture of the M3D system and editor, and then we present the Mu3D protocol and discuss how we assure database consistency. We also demonstrate analytically that consistency is achieved. Finally we present and discuss some ISDN traffic analysis result and conclusions.

## 2 SYSTEM ARCHITECTURE

The M3D system is designed as a fully distributed, multi-user, multi-layered system connected by the communication network. From the user’s point of view, the M3D system is composed of a set of different applications that are replicated in each workstation on each site. Sets of such workstations are connected through high to low band network to form the system. Figure 2.1 shows the general configuration of the M3D system.

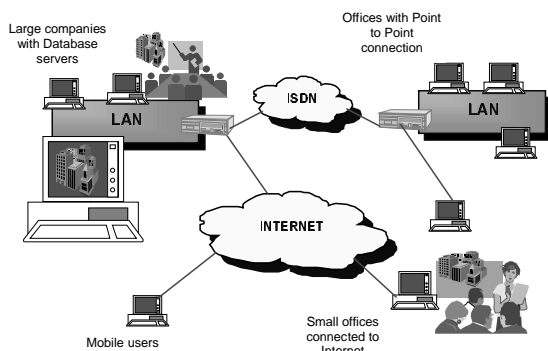


Figure 2.1: General Configuration of the M3D system

The M3D system follows the peer-to-peer network communication model. The layers of the M3D system and its corresponding layers to the OSI layers are shown in Figure 2.2. The cooperative support layer is application and network independent. The hosts participating in a session have the same set of resources and replicas of the applications.

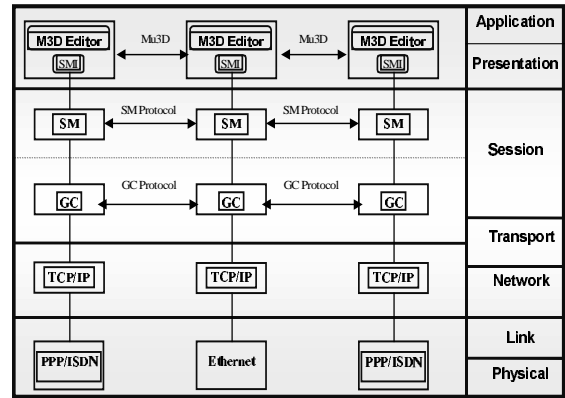


Figure 2.2: The layered structure of the M3D system and the corresponding OSI layers

### Local Structure

There is an instance of the structure depicted in Figure 2.3 in each site. Session control is achieved by a distributed protocol executed among the SM entities. These entities have the same set of capabilities. Some specific attributes may be given to just one of the session managers in order to perform special tasks such as permission functions given to the SM located in the conference initiator site.

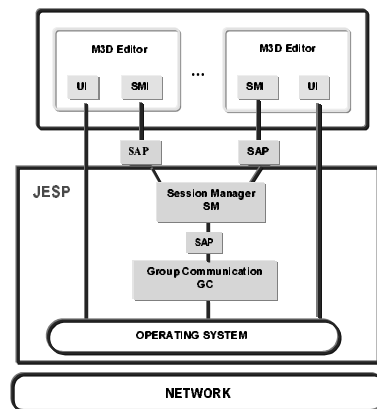


Figure 2.3: Local structure of the Editor and JESP

Each distributed application makes use of a specific protocol to exchange messages. Most of the messages contain *user events* produced through one or more input devices available at the user interface. These local events are encapsulated in messages and sent to the other sites through a service request made to the SM entity. The model in Figure 2.4 contains two I/O logical entities located at the application layer environment. One of them is the *User Interface (UI)*, which encompasses all the devices that collect local user events in the context of a particular application. The *Session Manager Interface (SMI)* structures the communication between the application and the SM layer.

### 2.1 The M3D Editor

The M3D Editor is a 3D distributed virtual environment (DVE) editing tool that immerses several users in the same world space

(Figure 2.1). Collaborators may interactively add, modify, or delete objects, change environmental attributes, and navigate in the common environment. The system is oriented to real-time WYSIWYG three-dimensional editing. It provides support for virtual **scene construction** where 3D object models of different formats and different standard commercial modelling packages may be input to construct the 3D environment.

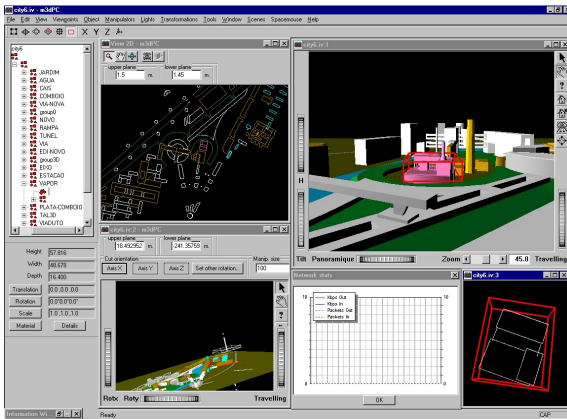


Figure 2.4: The M3D Editor (PC-Windows version).

The major entity in the editor is a VRML or OpenInventor 3D scene. A scene is a generic 3D object hierarchy, populated with typical OpenInventor [Wern94]/ VRML nodes [Ame96].

The editor is being developed using the TGS/OpenInventor toolkit as the 3D-development library. It runs in two platforms SGI (X11/Motif) and Windows machines.

### 2.1.1 Major Functions

The major function of the M3D Editor is the capability of editing an architectural design. It should have two modes: on-line cooperative editing mode and the off-line editing mode.

By using the M3D Editor, the chief architect can chair an on-line cooperative working session to achieve the tasks defined in the new business process, especially for integration and aggregation of design work from different specialists. Working session may also be held between the engineers and the internal architectural design team. The major editing functions are divided in several groups. They are operations and management for a scene and objects, for scene environments, and viewing management, etc.

**Object Management:** This group of functions is applied to a scene and objects (in VRML terms). A scene can be stored locally or in an HTTP server. The editor provides a relatively rich set of operations on the objects within a scene. New objects can be added, modified and removed from the scene. Their visual properties can be also defined and modified.

**Light and Rendering Management:** These groups are manipulation of environment properties of the scene. These properties are not part of the scene or objects. They will help to perceive the design work better in the virtual 3D environment.

**Text and Window Management:** These groups of functions provide viewing assistance to the user. Virtual camera

manipulation is within this group. When a window is created, there is a virtual camera associated with it.

### 2.1.2 Interactive Response

Since interactive cooperative design of a 3D virtual world usually involves a great amount of data to be manipulated and visualised, to minimise the network traffic becomes vital. This affects the feasibility of the whole M3D system of its multi-site cooperative design capability. Therefore, the interactive response time is a crucial requirement that the M3D Editor must meet.

The shorter response time requires the transmission of small data units to avoid long latency times. Another requirement to the M3D Editor is to have a uniform way to control the consistency of common designed data among the member hosts.

Our solution to meet this crucial requirement is two folds. On one hand, it is to implement a totally replicated in-memory database in each site. On the other hand, small size of messages will be sent via the network to represent the user modification.

## 2.2 CSCW Platform: JESP

The editor group communications are based on a communication and control platform for cooperative environments, the Joint Editing Service Platform (JESP). JESP is a generic groupware [Alme95] support platform. It was designed to provide two types of service to cooperative applications: group communication and session control. These are made available through a common service interface, a Computer Supported Collaborative Work system.

The group communication services hide the point-to-multipoint configuration from the application. The platform's session control mechanism supports the application's inclusion and control in cooperative environments. It includes a session control module (SM) and a group communication module (GC). The SM module is responsible for the tasks required by a cooperative working session such as service multiplexing, data consistency control, member admission, dynamic groups (early leaving and late admission), etc. The GC module provides the point to point or point to multi-point communication. Both modules correspond to the OSI session layer and partially to the transport layer.

## 3 REPLICATED DATABASE

A challenge in a multi-user 3D environment is to maintain consistent state among a large number of workstations distributed over a wide-area network [Awer97a, Awer97b]. Since three dimensional rendering at interactive rates requires fast access to the geometric database, the scene data are replicated on every site. Each replica of the program has its own in-memory, local database. The local database is a complex structure that stores all the required information to display the 3D object in the screen and to allow user interactions. Our database topology is a tree and is closely related to the VRML file format.

Whenever any entity changes its state, an appropriate update must be applied to every copy of the database in order to maintain consistency.

### 3.1 The Mu3D Protocol

In order to send the modifications of the scene to the peer application replicas during a running working session, an application layer protocol, the Mu3D Application Layer Protocol, has been designed. This protocol is an essential element to support the full multi-user interactions in a shared VRML environment. Inside the editor there is a sub-module in charge of the communication with the lower layer within the 3D-design application module called Session Manager Interface module. It encapsulates the Mu3D protocol information into a lower layer message. This message protocol, following the JESP nomenclature [Alme95], is the SMI-to-SMI protocol. This SMI-to-SMI protocol governs interaction within the common 3D scenario and it is implemented through the exchange of protocol data units (PDUs) among peers using well-known Services Access Points provided by the JESP platform.

Local changes are sent to the other replicas encapsulated in small "update" messages. These message formats are defined by the peer-to-peer Mu3D protocol (see Table 3-1). By this design strategy, each application replica on any workstation participating in a cooperative design session is able to modify scene parameters. From the point of view of the Editor, its "cooperative" job involves two major tasks:

1. Create the event locally: This task is carried out by the user interface to modify the scene nodes. Right after the local modification has been finished, the SMI sub-module has to encapsulate the new values into a Mu3D message and transmit it to the JESP platform.
2. Recreate remote events: Once the local JESP replica receives a message from its upper level, it sends it to the other remote JESP replicas. Each of them creates an event (TELE\_EVENT) to the their local editor. After the message is delivered to the application, it will interpret the message and modify the referenced node parameters.

#### 3.1.1 Structure of the Mu3D Data

The architectural design usually contains massive amount of data. To reach the interactive cooperative modification of the design work, we have to choose a workable strategy. The major strategy to make all the operations with reasonable interactive response time is to send as less data as possible via the network. By this strategy, we decide only to send the modification data of specific scene nodes and recreate the events locally. The Mu3D fields are designed to allow the editor to specify the user events in small messages. The current message size, counting JESP and TCP

headers varies between 150 and 220 bytes<sup>2</sup>. In our current test, we send the new node values coded in plain text for debugging purposes. This can be drastically reduced by sending binary data, which was not implemented yet to avoid subtle *endianess* problems among SGIs (*big endian*) and PC versions (*little endian*).

There are two main fields in the data unit: the EVENT (Mu3D protocol command), and the EVENT DATA (arguments).

EVENT	SCENE ID	OBJECT CLASS	OBJECT PATH	DATA LENGTH	OBJECT DATA
EVENT DATA					

Figure 3.1: Structure of the Mu3D Data

There are currently nine types of events defined in the M3D Editor. They are depicted in Table 3-1.

As shown in Table 3-1, some event types have been defined closely related to VRML entities such as nodes and scene graph. A *node* is the basic building block used to create 3D-scene database in both formats. Each node holds a piece of information, such as a surface material, shape description, geometric transformation, light, or camera. All 3D shapes, attributes, cameras and light sources present in a scene are represented as nodes.

An ordered collection of nodes is referred to as a *scene graph*. A *path* of a node is a chain of the nodes starting from the root down to a particular node. The indexing of nodes among its siblings follows a left to right order with the left most as number 0. The indexing of node uses the level of its depth in the graph from root and all the indexes of the nodes on the path<sup>3</sup>. The ObjectClass are class IDs in the VRML or OpenInventor scene graph. They are LIGHT, TRANSFORM, TEXTURE, FILE, CAMERA, etc. The ObjectPath is the string representation of an OpenInventor *Path* object. It is used for locating a node in the scene graph to apply certain action, for example, to delete an object from the scene. The ObjectData is a buffer that can contain any kind of data structure. The majority of the object data has been defined as strings and may be a location of URL of the files that can be included in a scene or the string representation of Node fields.

### 3.2 Database Consistency

When replicated data are updated, care must be taken to ensure that the updates occur in same controlled order at all replicas. Otherwise, local copies can become inconsistent. In an environment where no global ordering is guaranteed (JESP

<sup>2</sup> Early versions of the protocols had bigger message sizes but they were reduced optimising the headers and path representation. The average size in the current version is 130-150 bytes per application layer message.

<sup>3</sup> Each replica of the application maintains the state of remote selections (pointers to the actual object) to avoid inconsistencies that can appear when a user deletes the sibling of an object selected by another user.

platform only assures source ordering), this may be done by sending an update message and waiting for confirmation from every remote site [Birm87]. This synchronisation scheme means that the frequency of local updates is limited by the round trip time through the slowest path. Round trip values in ISDN and Internet may vary between 60 to 1000 milliseconds. Due to the highly interactive nature of the editor, these times are unacceptably high.

Event	Object Class	Object Data	Action
ADD	FILE	Filename	Insert a scene stored in a file in the given path.
	AVATAR	Filename	Insert an avatar in the database.
	URL	Url	Insert a VRML or OpenInventor scene using the HTTP protocol.
	ANYTHING	VRML class name	Create and insert a new node in the given scene path.
REMOVE	ANYTHING		Remove the given node from the database.
	LIGHT	VRML light node	Remove a light from the group of active lights.
MODIFY	AVATAR	Transform	Modify the position of the avatar.
	NAME	String	Change the name of the given node.
	TRANSFORM	Transform values	Modify the transform of the selected object.
	MATERIAL	Material values	Modify material properties.
	LIGHT	Light values	Modify the parameters of the light.
CLIPBOARD	COPY	Group node	Copy the node into the "distributed" clipboard.
	PASTE	Group node	Paste the copied node into the scene.
SELECT	GROUP	Group node	Mark the object (sub-tree) as selected by the sender.
SELECT ACK	USER	Boolean	A site acknowledges, in a point to point connection, the selection of the "requested" object. The ACK may be negative to indicate "rejection".
DESELECT	GROUP	Group node	Release the sub-tree (the user is known by the JESP platform)
TELE INIT	USER	User ID	To initialise an M3D working session by a session initiator and inform all other participants about its presentation. We decide to use an avatar to represent each participant visually in the scene.
TELE NOTIFY	USER	User ID	The answer message from a new participant. It has the same data contents as the user init event.

Table 3-1: Structure and events of the Mu3D message.

SEQEMBEDSEQREFREFTo ensure higher update rates our solution is to implement a scheme similar to the *causal broadcast primitive* (CBCAST) proposed by Birman and Joseph [Birm87] for supporting distributed computations in *fault-tolerant process groups*. The CBCAST primitive is used in the scheme to enforce a delivery ordering with minimal synchronisation.

Before defining temporal and causal relationships of a message, we introduce the *flow* concept. This concept was used in a very similar fashion by Yavatkar and Lakshman in [Yava94].

### 3.2.1 Flows

We can think of our collaborative system as a multiple flows "conversation". If we analyse the messages from a user during a short period, we will realise that they are updates to the same object. This occurs because we force users to select an "object"<sup>4</sup> before he is able to edit it. This policy is defined by the "selection" policy (see following section). In other words, the mechanism assures that the updates to branches of the scene tree are mutually exclusive.

Our concept of flows is analogous to "critical regions" in mutual exclusion algorithms. The important feature of these algorithms is that if one process is executing in its critical section, no other process is allowed to execute the same critical section. Similarly, our definition of flows ensure that while an editor replica is modifying a branch of the VRML scene tree, no other replica is allowed to modify nodes in the same branch.

To ensure the mutual exclusion feature we force the editor to acquire the "lock" by first sending the select Mu3D message and wait until an "ACK" is received from every session member. This procedure is, again, analogous to the entry section procedure in mutual exclusion algorithms.

**Definition 1:** If a lock to an object  $X$  is acquired by user  $i$  ( $L_X \in U_i$ ), we say that a new flow  $f_{X,i}$  was established ( $f_X$  for short) and  $f_X \in F$  where  $F$  is the set of existing flows in a running collaborative session.

A flow is actually defined by the existence of its corresponding lock. Once a lock has been released, the flow is also relinquished.

When collaboration involves communicating via a single or multiple flows, causal relationships among messages sent over the flows must be maintained to preserve the context in which a message is sent [Yava94].

#### Selection Policy

Our selection policy requires that the selection algorithm, which is applied to all the participating applications, preserves sequential ordering [Agra94, Atti94, Feke95, Lamp78, Yehu93]. Sequential consistency requires that all the data operations appear to have executed atomically, in some sequential order that is consistent with the order seen at individual processes. In other words, the same sequence can be repeated in a centralised shared-memory system.

By the definition of the Mu3D protocol in the above section, local copies will be updated by sending the changes encapsulated in *update* data packets. To maintain the consistency we must avoid possible conflicts that can appear when two or more users intend to modify the same *region* of the scene. The major solution we choose is to apply the locking policies and transaction-oriented operations. These policies are based on techniques coming from

<sup>4</sup> An object must be interpreted as the VRML sub-tree beneath a given group node.

distributed processing and database areas [Atti94, Awer97, Barr96, Feke95, Galli97, Glen81, Lync87, and Lync97].

We define the following *select policy*:

1. We allow a user to modify only selected objects. The selection of an object involves the “locking” of the VRML sub-tree under the selected node. During the period the object is selected, no other user can select such object (Figure 3.3).
2. An object can be selected only if no other member have previously selected any part of the object sub-tree. Once the selection is accepted by all members (one select generates N-1 ACK messages). If an ACK is not received in a given period of time (2 seconds in our implementation) then an error situation is assumed (disconnection or early leaving). This error will be detected soon by JESP architecture and communicated to the application.

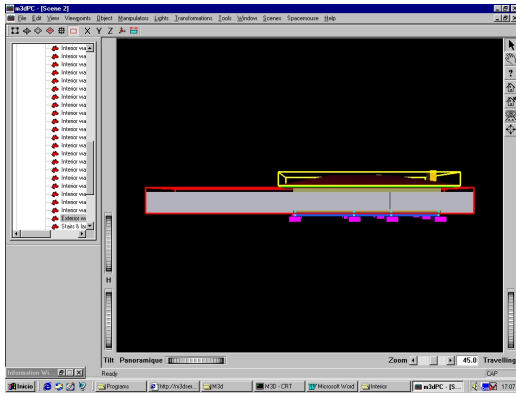


Figure 3.3: Local (lower, red bounding-box) and remote (yellow) selected objects

### 3.2.2 Temporal and Causal Relationships

Let  $F$  denote the set of flows in a collaborative session<sup>5</sup> and let  $M$  denote the set of messages sent over the constituent flows in  $F$ .

**Definition 2:** We define  $\pi$  (precedes) to be a transitive relation on  $F$  and  $start(f_X)$ ,  $end(f_X)$  the starting and ending time of flow  $f_X$  respectively, such that  $f_X \pi f_Y$  if and only if  $end(f_X) < start(f_Y)$ .

Definition 2 (and 3) is similar to Lamport’s “happens-before” condition [Lamp78].

**Lemma 1:** From the implementation of the selection policy and Definition 2, it is easy to prove that if object  $X'$  is a descendant of  $X$  or  $X = X'$  ( $X \subseteq X'$ ). If  $f_{X',i}, f_{X',j}$  occur in the same session, one of the following conditions holds:

1.  $f_{X',i} \pi f_{X',j}$

<sup>5</sup> Yavatkar call it “conversations”.

2.  $f_{X',j} \pi f_{X',i}$ .

**Definition 3:** We define  $\pi$  (precedes) to be a transitive relation on  $M$ , such that  $m_1 \pi m_2$  if and only if the following conditions hold:

1. Both  $m_1$  and  $m_2$  are sent by the same sender.
2. Both  $m_1$  and  $m_2$  are sent over flows  $f_i$  and  $f_j$  ( $f_i, f_j$ ).
3. The message  $m_1$  is sent before  $m_2$  is sent.

Definition 3 defines a partial order among all messages sent by the same sender, which is known as source ordering delivery in the field of distributed computing. The source ordering is assured by the JESP platform. The last can be demonstrated because JESP uses connection oriented TCP channels, what are defined as a “reliable FIFO channel,” which provides source ordered packet delivery (see for example [Lync97]).

**Lemma 2:** We can demonstrate from previous definitions that, in our implementation, messages that update the same object but belong to different flows with a precedence relationship between them, then the first messages was delivered before the second. If:  $m_i \in f_X$ ,  $m_j \in f_{X'}$  and  $f_X \pi f_{X'}$ , then  $m_i$  was delivered before  $m_j$ . It is simple to show that in order to “establish” the flow  $f_{X'}$ , all sites received the release message for  $f_X$  before “Acknowledging” the select message for the establishment of  $f_{X'}$ .<sup>6</sup>

Following Lamport’s definition of causality and sequential consistency [Lamp78, Lamp79]:

*The result of any execution is the same as if the operations of all processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program.*

The previous definition defines the “sequential consistency”. Informally speaking, a sequentially consistent memory [Feke95] appears to its users as if it was centralised. In other words, it requires that all the operations appear to be executed atomically, in some sequential order that is consistent with the order seen at individual processors (sites)<sup>7</sup> [Atti94].

In a distributed system, where updates to replicated database are sent using messages through some multicast channel, it is easy to demonstrate that if the message delivery mechanism respects “Lamport causality” and the sequential consistency is assured [Feke95]. Causal ordering can be defined as:

<sup>6</sup> It is worth to consider that all select, deselect and ACKs message belong to a special flow.

<sup>7</sup> This condition is similar to *serializability* from database theory.

When a message is delivered, the recipient has already seen any other message whose content could have been known to the sender at the time of sending.

As described in [Birm87], consider a broadcast  $m$  made a site  $p$  to update copies of a replicated variable  $x$ . Let this be followed by a broadcast  $m'$  to update copies of  $y$ . Although there is a potential causal relation between  $m$  and  $m'$  (because  $m \pi m'$ ), there may be no real causal relation between them. In this case, there would not be any reason to force the delivery of  $m$  before  $m'$ .

Exploiting the flow concept, we define a causal dependency relation  $\rightarrow$  among messages in a session as:

**Definition 4:**  $m_1 \rightarrow m_2$  if and only if at least one of the following two conditions holds:

1.  $m_1 \pi m_2$ .
2.  $m_2$  is sent over some flow  $f_X$  by sender  $S_1$  after receiving  $m_1$  over some flow  $f_X$ , with  $f_X, f_X' \in F$  and  $X' \subseteq X$ .

### 3.2.3 Correctness of the Implementation

To demonstrate the correctness of our database consistency mechanism, it suffices to show that the following conditions hold:

**Proof 1 (source ordering):** If  $m_i \pi m_j$  then  $m_i$  is delivered to their recipients before  $m_j$ . This condition holds because according to **Definition 3**,  $m_i, m_j$  were sent by the same site  $S_k$ .

They are delivered in the right order because the JESP platform defines a FIFO channel (TCP connection) between every editor replica. FIFO channels deliver messages in the same order they entered the channel.

**Proof 2 (mutual exclusion):** If  $m_i, m_j$ , with  $m_i, m_j \in f_X$ ,  $f_X \in F$  and is an active flow, then  $m_i$  and  $m_j$  were sent by the same site  $S_k$  (mutual exclusion). This condition is assured by Definition 2 and Lemma 1.

**Proof 3 (causal synchronisation):** If  $m_i \rightarrow m_j$ , then  $m_i$  is delivered to the recipients before  $m_j$ . This is the most important condition because it means we assure causal synchronisation, or in other words, the updates of distributed variables are "sequentially consistent". It suffices that one of the two conditions of Definition 4 holds:

- a)  $m_i \pi m_j$ : This condition is true because Proof 1.
- b)  $m_2$  is sent over some flow  $f_X$  by sender  $S_1$  after receiving  $m_1$  over some flow  $f_X$ . If  $f_X$  is the same active flow ( $f_X = f_X'$ ), both messages were sent by the same site  $S_1$ , by Proof 1 and Proof 2, the condition holds. If  $f_X \neq f_X'$  then

according to Lemma 1, one of the following conditions is true;  $f_X \pi f_X'$  or  $f_X' \pi f_X$ . Due to the mutual exclusion, if  $S_1$  sent  $m_2$  after receiving  $m_1$ , then  $f_X \pi f_X'$  and by Lemma 2, this condition holds.

### 3.2.4 Execution Examples

In this section, we present three executions that show how inconsistencies can appear if the system does not ensure causal synchronisation.

A conflict-free run of the system using the Mu3D protocol is depicted in Figure 3.4. This is normally the case, where due to the relatively low network roundtrip times are small compared to user interaction intervals. In the example Site 1 modified and released the lock before Site 2 sent a select message and started editing without waiting for any synchronisation or acknowledgement messages.

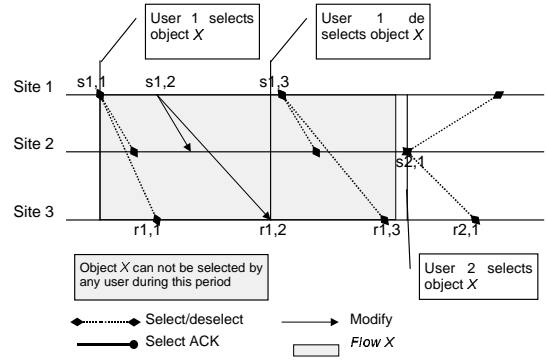


Figure 3.4: Normal execution with no conflict.

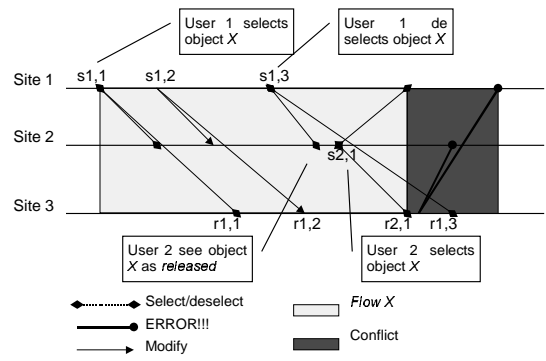


Figure 3.5: Execution with conflict and no causal synchronisation.

Although it seldom appears, a more realistic example is shown in Figure 3.5. In this example, the system does not provide any causal consistency mechanism. Site 2 received the deselect message from Site 1 ( $s_{1,3}$ ) and immediately selected the same object (message  $s_{2,1}$ ) before Site 3 received the previous deselect message from Site 1. This case may occur if packets travel between sites through different paths, and their roundtrip times vary noticeably. If Site 2 modifies its local copy before  $s_{1,3}$



Figure 4.4 and Figure 4.5 show the traffic pattern generated by object editing with visual manipulators and the Magellan 3D mouse respectively. The bandwidth control was also applied for 3D mouse events due to the high rate of generated events performed by the device (more than a thousand per second).

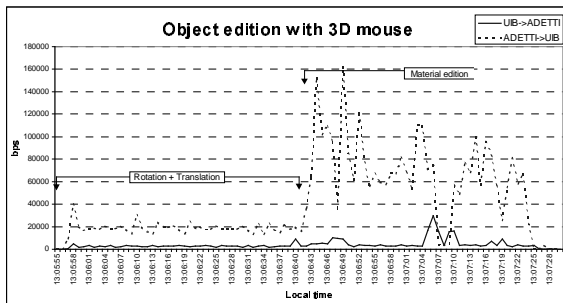


Figure 4.5: Transform and material editing traffic.

The required bandwidth for virtual navigation and object editing is approximately equal. Figure 4.5 shows a strange behaviour of the material editor. The traffic has exceeded the capacity of one ISDN channel (a second B-channel link was established automatically). The cause of this behaviour is similar to the one caused by the 3D-mouse device. The material editor module, which runs as an independent thread, generates one event for every insignificant colour change. The colour editing traffic pattern convinced us that the sub-sampling technique should be also used for every editing operation that can produce events in reaction to external input devices.

## 5 CONCLUSIONS

In this article, a protocol that collaborative editing of CAD design was presented. The development of the system is a challenge because no DVE system exists to our knowledge that allows full user interaction editing and offering at the same time fine-grained locking policies and bandwidth control techniques. The Mu3D protocol has shown suitable for interactive editing due to its low bandwidth requirements.

Although traffic can be further reduced by introducing sub-sampling "streams" in all user interaction, extensive tests of the editor showed us that a collaborative, fully interactive 3D-environment editing is achievable through both LAN and WAN network technologies.

Some useful concepts, such as flows and causal relationships were also defined. They served us to demonstrate analytically the correctness of the implemented database consistency techniques.

### Acknowledgements

The authors thank the M3D UIB team, the M3D Project partners and Miguel Dias and Sérgio Alves for their collaboration, efforts and new ideas.

## 6 REFERENCES

- [Agra94] D. Agrawal, J. L. Bruno, A. El Abbadi, V. Krishnaswamy. "Relative serializability: An Approach for Relaxing the Atomicity of Transactions." PODS '94. Proceedings of the Thirteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of Database Systems, pages 139-149.
- [Alme95] A. Almeida and C. A. Belo. "Support for Multimedia Cooperative Sessions over Distributed Environments." *Proc. Mediacomm'95*, Society for Computer Simulation, Southampton, April, 1995.
- [Ames96] Ames A. L., Nadeau D. R., Moreland J. L. The VRML 2.0 Sourcebook. John Wiley, New York (1996).
- [Atti94] Hagit Attiya, Jennifer L. Welch. "Sequential consistency versus linearizability." *Transactions on Computer Systems* Vol. 12, No. 2 (May 1994), Pages 91-122.
- [Awer97] Baruch Awerbuch, Leonard J. Schulman. "The maintenance of common data in a distributed system." *Journal of the ACM*, Vol. 44, No. 1 (Jan. 1997), Pages 86-103.
- [Barr96] Barrus J. W., Waters R. C., Anderson D. B. "Locales and Beacons: Efficient and Precise Support for Large Multi-User Virtual Environments." *Proceedings of the IEEE VRAIS'96 Conference*, IEEE Computer Society Press, Los Alamitos, CA, March 1996.
- [Birm87] K. P. Birman and T. A. Joseph. "Reliable Communication in the Presence of Failures." *ACM Transactions on Computer Systems*, Vol. 5, 1, February 1987, 47-76.
- [Brol97a] Broll Wolfgang. "Distributed Virtual Reality for Everyone – a Framework for Networked VR on the Internet". Proceedings of the IEEE Virtual Reality Annual International Symposium 1997 (VRAIS'97), IEEE Computer Society Press, 121-128.
- [Brol97b] Broll Wolfgang. "Populating the Internet: Supporting Multiple Users and Shared Applications with VRML." *Proceeding of the VRML'97 Symposium*, Monterey, CA, Feb 1997, ACM SIGGRAPH, 87-94.
- [Cad1] CAD Studio, "VRMLout for AutoCAD", <http://www.cadstudio.cz/indexuk.html>
- [Coul94a] Coulouris, G. and Dollimore, J. "Requirements for security in cooperative work: two case studies." Technical Report 671, Dept. of Computer Science, Queen Mary and Westfield College, University of London.
- [Dewa94] Dewan, P., Choudhary, R. and Shen, H. "An Editing-Based Characterization of the Design Space of Collaborative Applications." *Journal of Organizational Computing*, 1994, Vol. 4, pp. 219--239.
- [Fek95] Alan Fekete, Frans Kaashoek and Nancy Lynch. "Implementing Sequentially-Consistent Shared Objects Using Group and Point-to-Point Communication." *In the 15th International Conference on Distributed Computing Systems (ICDCS95)*, p.p. 439-449, Vancouver, Canada, May/June 1995,

- IEEE. Abstract/Paper. Also, *Technical Report MIT/LCS/TR-518*, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA, June 1995.
- [Funk95] Thomas A. Funkhouser. "RING: A Client-Server System for Multi-User Virtual Environments." *1995 Symposium on Interactive 3D Graphics*, Monterey CA USA.
- [Gall97] R. Galli, P. Palmer, M. Mascaro, M. Dias, Y. Luo. "A Cooperative 3D Design System." *Proceedings of CEIG97*, Barcelona, Spain, June, 1997.
- [Glen81] Glenn Ricardt and Ashok K. Agrawala. "An optimal algorithm for mutual exclusion in computer networks." *Communications of the ACM*, 24,(1):9-17, January 1981. *Corrigendum in Communications of the ACM*, 24(9):578, September 1981.
- [Hard96] J. Hard and J. Wernecke. *The VRML 2.0 Handbook*. Addison-Wesley, 1996.
- [Knis90] Knister, J. Michael; Prakash, Atul. "DistEdit: A distributed Toolkit for Supporting Multiple Group Editors." *CSCW '90 Proceedings*, ACM SIIGCHI & SIGOIS, L.A. Oct/90.
- [Lamp78] L. Lamport. "Time, clocks, and the ordering of events in a distributed system." *Communications of the ACM* 21(7), July 1978, 538-565.
- [Lamp79] L. Lamport. "How to make a multiprocessor that correctly executes multiprocess programs." *IEEE Trans. Comput. C-28*, 9 (Sept. 1979), 690-691.
- [Luo98] Y. Luo, R. Galli, M. Mascaro, P. Palmer, F. J. Riera, C. Ferrer, S. F. Alves, Real Time Multi-User Interaction with 3D Graphics via Communication Network, Proceedings of IEEE 1998 Conference on Information Visualization, July 1998, London.
- [Luo99a] Y. Luo, D. Sánchez, S. Alves, M. Dias, R. Marques, A. Almeida, J. Silva, J. Manuel , , B. Tummers (EDC), Ed. R.Galli, "M3D technical specifications", ESPRIT Project No. 26287 M3D, Deliverable 1.2, April 1999.
- [Luo99b] Yuhua Luo, Ricardo Galli, Antonio Carlos Almeida, Miguel Dias, A Prototype System for Cooperative Architecture Design, Proceedings of IEEE 1999 International Conference on Information Visualization, July 1999, London, pp. 582-588.
- [Lync87] Nancy A. Lynch and Mark Turtle. "Hierarchical correctness proofs for distributed algorithms." Master's Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, April 1987. Technical Report MIT/LCS/TR-387. Abbreviated version in *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing*, p 137-151, Vancouver, British Columbia, Canada, August 1987.
- [Lync97] Nancy Lynch. *Distributed Algorithms*. Morgan Kaufman Publishers, Inc. ISBN 1-55860-348-8. 1997.
- [Sall97] José Miguel Salles Dias, Ricardo Galli, António Carlos Almeida, Carlos A. C. Belo, and José Manuel Rebordao. "mWorld: A Multiuser 3D Virtual Environment." *IEEE Computer Graphics and Applications*, Vol. 17, No. 2, March/April 1997.
- [Wern94] J. Wernecke. *The Inventor Mentor*. Addison-Wesley Publishing Company, 1994.
- [Yava94] R. Yavatkar, K. Lakshman. "Communication support for distributed collaborative applications." *Multimedia Systems* (1994) 2: 74-88.