Architectures and Mechanisms to efficiently Maintain Consistency in Collaborative Virtual Environments

Cédric Fleury*
INSA de Rennes
IRISA, UMR CNRS 6074
Université Européenne de Bretagne, France

Thierry Duval†
Université de Rennes I
IRISA, UMR CNRS 6074
Université Européenne de Bretagne, France

Valérie Gouranton‡
INSA de Rennes
IRISA, UMR CNRS 6074
Université Européenne de Bretagne, France

Bruno Arnaldi§
INSA de Rennes
IRISA, UMR CNRS 6074
Université Européenne de Bretagne, France

ABSTRACT

Collaborative Virtual Environments (CVE) enable users to collaborate and interact together by sharing a common virtual environment. Ensuring the global consistency of a virtual environment is very important to provide efficient collaboration between users. However, users sharing a CVE may be scattered over different physical locations, so CVE systems have to guarantee the virtual environment consistency despite network issues such as low bandwidth or network latency. Absolute consistency is nearly impossible to achieve because it would prejudice the responsiveness of the system during user interactions. So, CVE systems have to deal with a trade-off between consistency and responsiveness of the system. This paper presents a detailed survey of architectures and mechanisms used to improve the consistency of a shared virtual environment. Architectures of CVE systems are studied according to their impact on consistency. Contrary to previous state of the art reports, which classify CVE systems according either to the network connections or to the data distributions, we choose to examine both of these two properties independently. This classification enables us to deal with the increasing number of hybrid architectures that mix different architectural choices to meet their requirements. Consistency maintenance mechanisms are also examined. First, a time synchronization must be achieved in order to enable users to perceive the same state of the virtual environment at the same time. Second, the virtual environment can be seen as a database shared by several users, so CVE systems must manage users' concurrent access to shared virtual objects. Finally, we discuss about how CVE systems can be improved to deal with performance constraints induced by the use of mainstream networks such as DSL Internet access and by security requirements of industrial users.

Index Terms: H.5.3 [Information Interfaces and Presentation (e.g. HCI)]: Group and Organization Interfaces—Computer-supported cooperative work; C.2.4 [Computer-Communication Networks]: Distributed Systems—Distributed applications; I.3.7 [Computer Graphics]: 3-Dimensional Graphics and Realism—Virtual reality

1 INTRODUCTION

One of the main goals of Collaborative Virtual Environments (CVE) is to enable users to work together in a natural way and to provide them a truly interactive experience. Generally, each user uses his own computer to have individual interaction capabilities or to meet the others if they are not located at the same geographical place. So, this collaborative work must be achieved over a local area network (LAN) or a wide area network (WAN) between the users' computers that we will call "nodes". For example, in the case of the French ANR project Collaviz[1], remote experts have to analyze together scientific data using an Internet connection through a secured proxy. Such network connections have a strong impact on the consistency of the shared virtual environment because they delay the transmission of the virtual environment changes. While some users of a CVE can interact through immersive devices, powerful computers, and high-bandwidth network connections, some other users can share the same CVE trough simple workstations and low-bandwidth network connections. Even if not powerful enough computers or low-bandwidth network connections put some users at a disadvantage, these users have the right to share the same state of the virtual environment than the other users. Ensuring the CVE consistency is the best way to make possible an efficient collaboration between users because it can avoid conflicts between several user actions or misunderstandings when users perform collaborative tasks.

To define the consistency of a CVE, Delaney et al. [7] explain that a CVE must be considered as a distributed database with users modifying it in real-time. Consistency maintenance consists in ensuring that this database is the same for all the users, i.e. users observe or interact with the same data. The consistency of a CVE can be characterized by the following three criteria:

- **synchronization:** (1) time synchronization: an event (state modification of a shared virtual environment) should happen simultaneously on all the nodes. (2) spatial synchronization: CVE objects should have the same location at the same time on all the nodes.
- **causality:** events order must be the same for all the users.
- **concurrency:** conflicts can occur when users change the same parameter of a virtual object at the same time. These conflicts have to be managed to avoid that users make their own modifications and have inconsistent states of the CVE.

Consistency is directly linked to system responsiveness. Responsiveness can be defined as the time needed by the system to respond to user actions. Responsiveness during interactions can be quantified by the system latency, i.e. the time between a user action and the system response. This latency is often due to the transmission time over the network and to the processing delays of the events. Improving the consistency of a CVE can increase latency during interactions and vice versa. So, CVE systems must reach a trade-off between consistency and system responsiveness.

Latency is especially a problem for highly interactive applications. Delaney et al. [7] present several values of latency found in the literature. It seems that no consensus has been found about these values. The maximum latency values fluctuate between 40 and 300 ms to ensure real-time interactions, and a maximum latency of 100 ms seems sufficient for most of the applications. These values also depend on the jitter (the variation of the latency) of the system. Roberts et al. [22] explain that jitter has a more significant impact

*e-mail: cedric.fleury@irisa.fr
†e-mail: thierry.duval@irisa.fr
‡e-mail: valerie.gouranton@irisa.fr
§e-mail: bruno.arnaldi@irisa.fr

[1] www.collaviz.org

on user performance than latency. It would be better to have a quite high and constant latency (i.e. with a low jitter), rather than a lower latency with a higher jitter.

In this paper, Section 2 describes the different architectures of CVE systems and their impact on consistency and system responsiveness. Then section 3 examines the consistency maintenance mechanisms used in CVE systems. Finally, section 4 concludes and discusses what can be improved to maintain the consistency in CVE systems that use mainstream networks and need secured network connections.

2 System Architecture

Consistency and performance of a CVE are highly correlated with its system architecture. For example, some architectures maintain a strong consistency of the CVE but introduce latency during interactions. In contrast, other architectures accept a few inconsistencies but offer a better responsiveness during interactions. Previous state of the art reports classify CVE systems according either to how the nodes are connected together [8, 13] or to how data are distributed on these nodes [17, 9]. Although these two characteristics are often interrelated, we chose to examine both of them independently for our classification: the kind of network architecture (part 2.1) and the kind of data distribution (part 2.2). This classification choice is motivated by the fact that more and more hybrid solutions mix different architectural choices to meet their requirements. As different communication protocols can be used to communicate information during a session, we chose this third criterion to complete our classification (part 2.3).

2.1 Network Architecture

A CVE system is usually made up of several interconnected nodes that can be geographically scattered. Each node can communicate with the others through three main data transmission methods:

- *unicast:* transmission from one node to another one.
- *broadcast:* transmission from one node to all the others.
- *multicast:* transmission from one node to a subset of other nodes.

Delaney et al. [8] distinguish two basic network organizations used for CVE systems: the peer-to-peer architecture and the client/server architecture. They also introduce hybrid architectures that combine these two solutions.

2.1.1 Peer-to-peer architecture

The peer-to-peer architecture enables fast communications between pairs of users, because events are transmitted directly from one node to another one (see Figure 1). So, it enables a few users to have strong synchronization, and consequently closely coupled interactions. However, increasing the number of users will increase hugely the number of messages transmitted on the network, especially when an *unicast* transmission is used (if the session contains N members, a node has to send N-1 messages to transmit one event). Consequently, it is difficult to contact all the nodes at the same time to transmit virtual environment changes. So, time synchronization and global consistency of the CVE may be difficult to ensure. This kind of architecture appeared with the first CVE systems (Reality Build for Two [2] that connects only two computers, MR Toolkit [24]) and is used in military applications (SIMNET [5], NPSNET [18]).

2.1.2 Client/server architecture

This kind of architecture is based on a central server. All the nodes get connected to this server (see Figure 2). The central server manages the communication between different nodes and stores data. This kind of architecture enables the server to contact all the nodes at the same time. So, time synchronization and CVE consistency

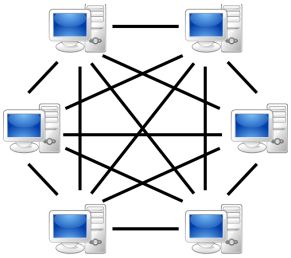

Figure 1: peer-to-peer architecture

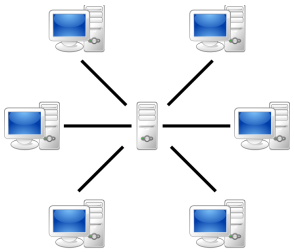

Figure 2: client/server architecture

are easier to maintain than with the peer-to-peer architecture. However, when two users want to interact together, all the communications have to pass through the server, which increases latency during interactions. When the number of users increases, a bottleneck can occur on the server due to too many communication requests, so all the communications can be slowed down. For example, a client-server architecture is used in: RING [11], BrickNet [25], and ShareX3D [14].

2.1.3 Hybrid Architecture

To overcome the previous limitations, some systems propose an hybrid network architecture that uses both peer-to-peer connections and one or several servers. For example, it is possible to speed up communications between nodes by using peer-to-peer connections, and it is possible to maintain a better consistency with a server.

In SPLINE [26], several servers share up-to-date information (messages, events, etc.) with peer-to-peer connections between these servers (see Figure 3). At the beginning of a session, the session manager connects nodes to one of these servers, then nodes only communicate with their assigned server. This solution avoids the bottleneck on the server when the number of users increases, and it makes it possible to easily connect nodes with slower or secured connections. Indeed, each server can perform additional processing such as compression or communication with a specific protocol. However, the use of too many servers can increase the system latency and the load of the servers.

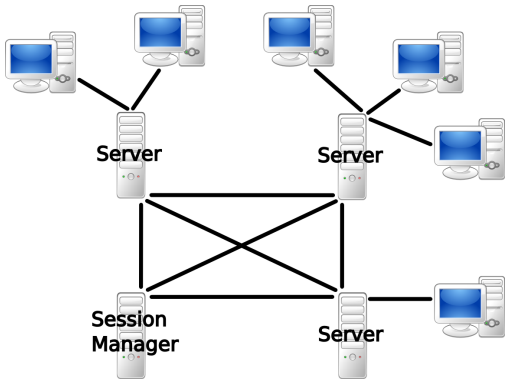

Figure 3: several servers using peer-to-peer connections

Anthes et al. [1] suggest another hybrid architecture to facilitate collaboration between nearby users (according to their location in the virtual environment) by reducing the latency between them. Users are connected to the CVE through a server. When some users come closer together in the virtual environment, temporary peer-to-peer connections are established between these users to increase the virtual environment consistency (see Figure 4).

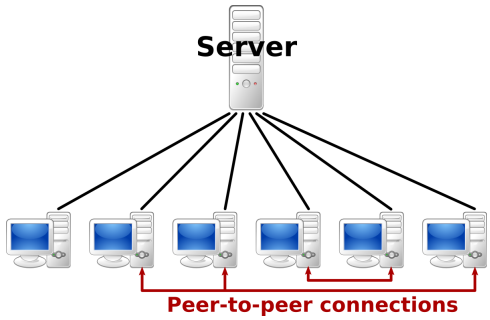

Figure 4: temporary peer-to-peer connections between close users

The OpenMASK platform [19] architecture is also an hybrid one. It uses peer-to-peer connections to send updates and events to nodes. It also uses a server to manage the identification of virtual objects and to add dynamically nodes during a session.

CAVERN [16] is based on a generic interface that defines the network connection and the data storage. Each node implements this interface to make possible a communication with all the other nodes. This node can act as a client or as a server, so any kind of network architectures can be implemented with this framework.

2.2 Data Distribution

As stated by Macedonia et al. [17], the location choice of the virtual environment data (i.e. geometric data, textures, etc.) is a critical decision when designing a CVE system. We must not only determine which computers store this data, but also which computers execute the processing related to each virtual object. We distinguish three data distribution modes: the shared centralized world, the homogeneously replicated world and the partially replicated world (distributed world).

2.2.1 Shared centralized world

Some systems such as Vistel [28] use one database shared by all the nodes. All the data relative to the virtual environment are stored on a central server. Similarly, behaviors of the CVE objects are executed on this server (see Figure 5(a)). Users log on the server to join a session (it requires a client/server network architecture). When a user wants to modify an object, his node sends a request to the server. The server processes the modification request, then transmits the up-to-date state of the object to all nodes, including the one that asked for this modification (see Figure 5(b)).

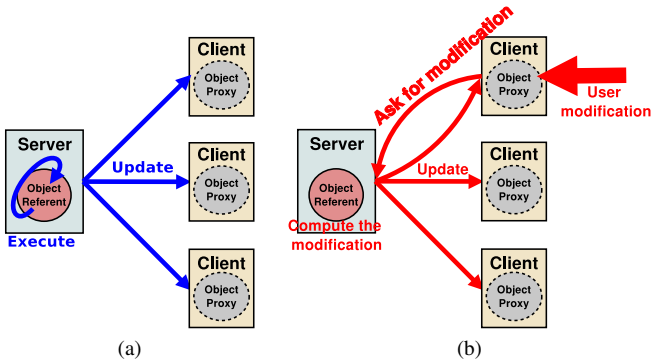

Figure 5: (a) executions of object behaviors and (b) object modifications in a shared centralized world

This method ensures consistency between all the nodes and avoids data replication, but it has two main drawbacks:

- During interactions, latency can increase when transmission delays occur between the clients and the server. Indeed, each modification request must pass through the server before returning to the user (see Figure 5(b)). Users can get annoyed with this lack of responsiveness during interactions.
- With many users, a bottleneck can appear on the server because it has to send updates to all the nodes at the same time (especially with *unicast* connections).

2.2.2 Homogeneous replicated world

This kind of data distribution is used in many CVE systems (SIMNET [5], MR Toolkit [24]). All the nodes are initialized with the same database that contains all the information about the virtual environment (terrain, geometric models, textures, object behaviors, etc.). Data can already be present on the node when the user logs in (such as in most of the video games). Otherwise, data must be replicated from a server, or from the other nodes already connected to the session. During the session, the database evolves independently on each node and all object behaviors are executed locally (see Figure 6(a)). Additionally, a synchronization mechanism can be used to control executions of object behaviors on each node. To maintain consistency, only object modifications and some special events (collision between two objects, etc.) are transmitted on the network in order to enable all nodes to update their database (see Figure 6(b)).

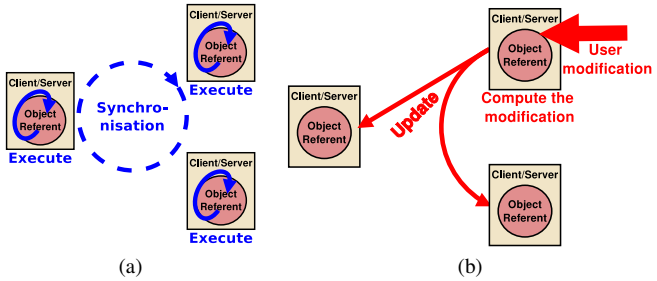

Figure 6: (a) executions of object behaviors and (b) object modifications in an homogeneous replicated world

This data distribution has two main advantages:

- The number and the size of messages transmitted on the network are really small because only update messages are sent.
- The latency is very low during user interactions. In fact, virtual object modifications are performed locally before being sent to the other nodes by using update messages.

However, data replication also has drawbacks:

- Data replication can introduce inconsistencies between users' virtual environments because of delays or losses during the transmission of update messages.
- Additional mechanisms must be provided to manage the concurrent access on each node. Indeed, a user can perform a local modification of an object, but modification conflicts can only be checked when the changes are transmitted to the other nodes.
- This solution is not really appropriate for very large data sets such as scientific data or complex CAD models, because each node may manage its own large database.
- This approach is not really flexible, especially if users want to add new objects in the initial database.

2.2.3 Partially replicated world (or distributed world)

Some CVE systems choose hybrid solutions between totally centralized and totally replicated data distributions in order to avoid the drawbacks of these two solutions. These hybrid solutions distribute the data and their processing among the nodes. It reduces

the necessary resources and eases the consistency maintenance. In the literature, these hybrid solutions are called partially replicated world or distributed world. We present some hybrid solutions that seem particularly interesting.

To avoid the duplication of all the data on each node, data can be distributed on the network among these nodes. So the database can be seen as a shared and distributed memory. In DIVE [10], when a new user joins a collaborative session, only the necessary objects are replicated on his node instead of downloading the whole data of the virtual environment (see Figure 7). However, if this user needs other objects during the session, they must be dynamically downloaded.

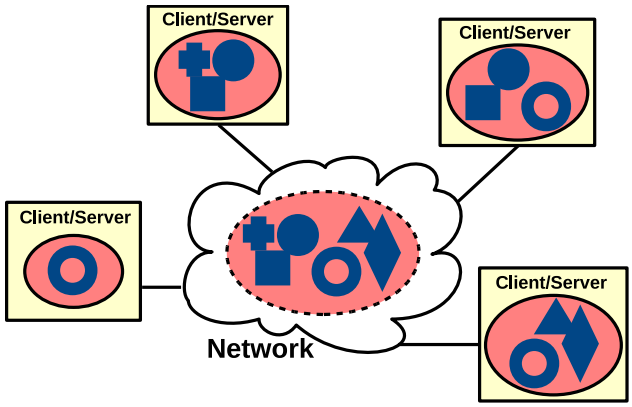

Figure 7: Data partially replicated on each node

DIVE uses peer-to-peer connections to manage communication between nodes (transmissions of messages or object data when needed). DIVE also uses a server to backup the data distributed all over the nodes, in order to save the state of the virtual environment when a user logs out, and to restore this session later. This method makes it possible to share CVE between many users and to share a very large amount of data without necessarily duplicate this data on each node. However, dynamic transmission of data and consistency maintenance induce a high cost of communication between nodes. When an object changes, update messages must be sent to all the nodes even if they do not own this object. Indeed, a node cannot know which node owns the modified object.

The main difficulty of this partially replicated world is to dynamically download data without disturbing user interactions. According to Lee et al. [15], two solutions can be used:

- The prioritized transfer: this technique consists in selecting first the objects that are in the user field of view, and transferring these objects using level of details (LODs) or multiresolution techniques. This solution maximizes graphical fidelity of the virtual environment as well as interactive performance by mediating graphical details and transmission overhead of the objects.
- Caching and prefetching the data that users will probably need: it makes data immediately available when users ask for it. However, this solution must predict efficiently which objects will interest first a user to define a loading priority. Generally, the distance between users and objects is used to determine this priority, assuming that users will be interested first by the closest objects. Other solutions suggest to add the moving direction of users or to use the objects type to determine the users' scope of interest. However, this prediction is difficult to achieve with lots of objects.

To reduce the cost of communications for the updates, BrickNet [25] uses a server to keep information about the shared objects: access rights, ownership, etc. It uses a client/server architecture: the server manages communication between nodes. Object behaviors are executed on each node, and the server keeps a list of nodes that share this object. When a user wants to modify an object, his node must first ask the server for the modification rights on this object (only one user can modify an object at the same time). When its request is granted, it can modify locally this object. Then the new state of the object is transmitted to the node that owns this object, using a list of object owners on the server. With this approach, the server makes it possible to ease the consistency maintenance and to manage concurrent access to objects.

In OpenMASK [19], data of the virtual environment are replicated on each node. Each object behavior is executed only on one node. To achieve this, OpenMASK [9] uses a referent/proxies paradigm. The referent is assigned to a particular node and defines the object behavior. On the other nodes, proxies are created to represent the object. A proxy provides the same interface as the remote referent (same inputs, same outputs, etc.). However, there is no processing done locally and the proxy is driven by its referent (see Figure 8(a)). The OpenMASK kernel [19] maintains the consistency between the referent and its proxies. When a user manipulates an object with the referent on his node, first the object is modified locally, second an update is sent to all the other nodes (see Figure 8(b)). If the referent of the manipulated object is not on his node, the object modification is computed first on the remote node that owns the referent. Then this node transmits updates to all the nodes, including the node that asked for the modification (see Figure 8(c)). In this second scenario, transmission delays on the network can introduce latency during interactions. However, this solution makes it possible: (1) to combine the processing power of all the nodes, (2) to ensure a better consistency of the virtual environment, and (3) to manage implicitly the concurrent access to the objects (in a first step, only the referent can be modified).

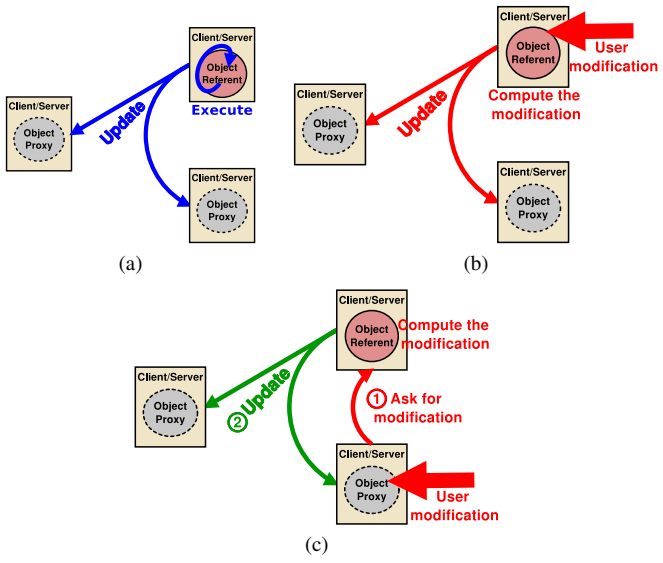

Figure 8: (a) executions of object behaviors, (b) direct modification of an object through its referent, and (c) modification of an object through on of its proxies in OpenMASK.

CAVERN [16] also uses a kind of referent/proxies paradigm for data distribution. Each object is defined by a "local key" on one node and "remote keys" on the other nodes. All these keys are linked together over a communication channel, so any modification of one key is automatically propagated to all the others.

2.3 Communication protocols

Several protocols can be used to communicate the information needed to maintain consistency between the nodes of a collaborative session. A protocol describes how the nodes will communicate on the network. A network connection can be decomposed in three layers: the network layer, the transport layer, and the application layer. Usually, protocols for CVE systems are defined either in the application layer or in the transport layer. The most commonly used network layer protocol is IP (Internet Protocol).

2.3.1 Classical Protocols

TCP (Transport Control Protocol) and UDP (User Datagram Protocol) are the most commonly used transport layer protocols. TCP ensures reliable transmissions between two nodes using a system of acknowledgment and retransmission. TCP can only achieve *unicast* transmissions. On the contrary, UDP sends data in a non-connected mode: there is no way to verify the correct reception and the correct order of the packets. However, it can be useful to quickly send data to several users (using *broadcast* or *multicast* transmissions). Indeed, a new packet can be sent without having to wait that all the previous packets have been received by all the concerned users. To solve the reliability problems of UDP, SPLINE [26] and SIMNET [5] send update messages with the whole state of the object (not only the changes). If some messages are lost, an up-to-date object state can be restored when a new message arrives. So there is no need to retransmit lost packets.

According to Delaney et al. [8], the choice of the transport layer protocol depends on the kind of interactions in the CVE. For punctual interactions that involve a transmission of a small amount of information or an irregular transmission, it is better to choose UDP. On the contrary, for persistent interactions that require a transmission of a large amount of information or a regular transmission, the TCP protocol is most commonly used.

2.3.2 *Multicast* oriented Protocols

The first CVE systems have chosen to use transmissions based on *unicast* or *broadcast*. For example, MR Toolkit [24] uses *unicast* transmissions between nodes widely distributed over the Internet. However, these solutions are not efficient when there are a lot of users. As in SIMNET [5], *broadcast* transmissions may be problematic in the case of many nodes connected to the network: it would submerge the network with unwanted data. Consequently, researchers have proposed several solutions using *multicast* protocols. To achieve *multicast* transmissions, a solution is to use the *multicast* IP network layer, which enables to send messages simultaneously to many recipients (on a non-reliable way). According to Delaney et al. [8], this solution has several drawbacks:

- it can be difficult to implement efficient *multicast* on a point-to-point medium.
- many Internet routers are not able to support *multicast*.
- the number of user groups may be limited because of various issues such as addressing, congestion control, or administration.

To overcome these problems, network overlays have been developed to provide IP *multicast* capability over networks that do not offer *multicast* capability at the network layer. DIVE [10] and NPSNET [18] use the "MBone" (*multicast* backbone) that creates a virtual network providing *multicast* with encapsulations of *multicast* packets in normal IP packets.

2.3.3 Virtual Reality dedicated Protocols

Other application layer protocols use the transport and network layer to optimize the transmissions of application-specific data. El Zammar [9] lists several application layer protocols dedicated to virtual reality applications:

Real-Time Protocol (RTP) provides network transport functionalities for applications transmitting real-time data such as video and audio streaming, or simulation data. It supports both *unicast* and *multicast* transmissions. Mauve et al. [21] propose a new protocol RTP\I that adapts RTP for interactive applications. RTP\I uses four types of packets to manage event communications, objects states transmissions, state changes, and state queries.

Virtual Reality Transfer Protocol (VRTP [4]) is designed to be a support for VRML (Virtual Reality Modeling Language) in the same way that HTTP is the HTML support. VRTP can be seen as an extension of the HTTP protocol. Its main goal is to meet the requirements of CVE, because HTTP is not sufficient to manage 3D interactive objects. The VRTP approach proposes that a node can take the role of a client, a server, or a peer. This node can be seen as a client examining databases of other nodes, as a server responding to requests from other nodes, and as a peer participating in a session with groups of nodes speaking through *multicast* channels.

Distributed Worlds Transfer Protocol (DWTP [3]) is a specific application protocol for sharing virtual environments over the Internet. It is based on standard protocols such as TCP/IP and UDP/IP. It makes it possible to transport several types of data: events, messages, files and data streams. Events are used to ensure the virtual environment consistency between the users. Messages are predefined events such as "join" or "leave" the virtual environment. A file can be a 3D scene description, an object geometry, etc. It requires a reliable transport. Data streaming transmits a continuous data stream such as audio or video. This data type does not require a reliable transmission. The DWTP concept is based on daemons and participants. The role of a daemon is to provide services to the participants:

- the reliability daemon: detects transmission failures (for unreliable protocols like UDP).
- the retransmission daemon: transfers lost data by using *unicast* connections.
- the world daemon: transmits the virtual environment content to new participants.
- the *unicast* daemon: extends the architecture for participants unable to use a *multicast* channel.

Distributed Interactive Simulation (DIS) comes from the SIMNET military simulation platform [5]. The DIS standard (IEEE 1278) defines specific messages called PDU (Protocol Data Units) which communicate specific information to the nodes. A few PDU are dedicated to user interactions with the virtual environment and the others are used to transmit various data and actions on the virtual environment such as object apparitions, weapons fires, explosions, etc. DIS is very specific to the military simulation context because message formats are dedicated to military simulation objects. In order to extend the scope of this protocol to non-military applications, a new version called High Level Architecture (HLA) has been proposed [6].

Interactive Sharing Transfer Protocol (ISTP [27]) has been developed in the context of SPLINE [26]. ISTP can also be used with other CVE systems. Each ISTP process acts as a client and as a server. To communicate information about virtual objects, ISTP is based on sub-protocols:

- the connection protocol: used to establish TCP connections between two ISTP processes.
- the object state transmission protocol: used to transmit the object states using *unicast* or *multicast* UDP connections.
- the data streaming protocol: used for the communication data streams such as audio (based on RTP).

2.3.4 Industrial Environment Specific Protocols

For some industrial uses, classical or VR dedicated protocols are not well adapted to modern security constraints. For example, ShareX3D [14] has to deal with firewalls that support only the HTTP protocol. So ShareX3D proposes to use the "long polling" technique for server-to-client and client-to-server-to-client communications (ShareX3D uses a client/server architecture). This technique is used to overcome the fact that HTTP does not allow requests from the server to the client. The "long polling" can be divided in four steps:

- The client sends a connection request to the server.
- As long as the server has no event to send to the client, the connection is kept open.
- Once the server has an event to send to the client, it transmits it with the open connection. Then the connection is automatically closed.
- The client immediately re-sends a new connection request to the server, and so on.

Establishing a new connection after sending each message is slow, so it may introduce latency during user interactions when events need to be sent with high frequency rates.

3 Consistency Maintenance Mechanisms

In addition to the system architecture and to the communication protocol, some mechanisms can be used to improve the virtual environment consistency. First, we present time synchronization mechanisms that enable each node to process CVE changes at the same time. Second, we examine different solutions used to manage the concurrent access to virtual objects.

3.1 Synchronization

Time is a fundamental element of a CVE. The notion of time can differ from one application to another. Delaney et al. [7] distinguish two different notions of time in CVE:

- **Absolute time:** the time of a CVE can be based on a clock periodically synchronized between each node, based on the coordinated universal time (UTC).
- **Logical or virtual time:** the time of a CVE can be based on a logical clock that is not precisely synchronized between each node as proposed by Jefferson [12]. This time can be seen as an ordered event sequence. When no new event occurs, it stays the same.

The relationship between time and consistency is very important. In a perfectly consistent CVE, all the users perceive the same state at the same absolute time. However, this perfect case can never happen because of network latency. Delaney et al. [7] list different solutions to improve consistency over time according to the required responsiveness for interactions.

3.1.1 Lockstep Synchronization

The lockstep synchronization used in RING [11] or OpenMASK [19] is the easiest way to ensure the consistency of a CVE. It consists in stopping nodes until all of them have processed the current simulation step. So, each node does not increment its logical clock until all the other nodes have acknowledged that they are ready for the next simulation step. This solution avoids roll-backs because it imposes that events are processed in the correct order. However, it guarantees consistency but not in real-time. If there are delays or losses during transmissions, the time spent to wait increases, and the system responsiveness breaks down. Furthermore, simulation steps are not necessarily constant, so the system jitter can be substantial.

3.1.2 Imposed Global Consistency

This technique consists in delaying the processing of both local and remote events. The delay is the same for all the nodes, its value is usually defined according to the maximum values of the network latency. This delay makes it possible to increase the probability that the remote events are received before processing all the events of a simulation step. This method makes possible a strong global consistency between all the nodes with an absolute clock. However, this solution introduces interaction latency which value varies according to the network characteristics. This latency is constant during the whole session.

3.1.3 Delayed Global Consistency

Contrary to the imposed global consistency, the goal of this technique is to maintain an asynchronous consistency. Users perceive the same state of the virtual environment, but not at the same time. Each event is marked with a "timestamp" (using a logical clock). Each node manages its logical clock. So the state of a CVE can be rebuilt locally with the events right order. However, this delayed consistency can disturb collaborative tasks (users may not perceive the same virtual environment at the same time).

3.1.4 Time Warp Synchronization

"Time Warp" synchronization proposed by Jefferson [12] is an optimist technique, which consists in processing each event as soon as it arrives. Events are also marked with a "timestamp". When an event is received with an older "timestamp" than the event that has just been processed, the mechanism must cancel the processing of all the events with a most recent "timestamp" (roll-back). Then it processes again all these events to catch up with the current time. Moreover, it must send messages to cancel incorrect messages sent during the deprecated execution (roll-back propagation).

This synchronization method makes possible low latency interactions. However, it can only be used when roll-backs happen rarely, because they are extremely annoying for users. Several systems propose to quickly display several key-frames during the re-execution of events to facilitate the users' understanding. Finally, this method requires to store the received events to re-execute them in case of roll-back.

3.1.5 Predictive Time Management

This method proposes to predict events and to send them on the network before they occur. This concept was first proposed by Roberts et al. [23] in the PARADE system to manage the consistency when users collaborate through a network with inherent latency. Obviously, this mechanism can not be applied to all the virtual environment objects because some objects are not predictable. Particularly, user actions are difficult to predict. For example, PARADE uses this method for collision detection.

Events are predicted locally, marked with a "timestamp" and sent to other nodes, where these events will be processed at the appropriate time (defined by the "timestamp"). This predictive management is interesting only if the time between the sending of the predicted event and its processing is higher than the network latency. Otherwise the message will arrive too late to be processed. If the prediction is false, the system needs to make roll-backs to resolve mistakes. To minimize roll-backs in PARADE, predicted events are sent "just-in-time" by using estimations of network delays: network delays are determined by the study of the RTT (Round-Trip Time) of the network.

3.1.6 Server Synchronization

In client/server architectures, the server can be used to synchronize events using a logical clock. In ShareX3D [14], the server maintains a "state number" for each object of the CVE. When the server receives a change for an object, it increases the "state number" of

this object. Nodes also maintain a "state number" for each object corresponding to the last update messages received for this object. When a node asks the server for new changes about an object (see "long polling" in 2.3.4), it sends the local "state number" for this object. If this "state number" is older than the server one, the server sends back an update message with the new "state number" value. Otherwise, the node is up-to-date and its request stays in standby.

This solution provides a simple way to ensure that nodes have the most recent object states, but it does not guarantee that all events will be received and processed by nodes. This is not an issue in ShareX3D because the server sends whole states of objects, so nodes can restore the state of an object with only one update message.

3.2 Concurrency control

The centralized data distribution ensures an implicit control of concurrent access to CVE objects, because this data can be changed only on the server. It is the same for systems that use a referent/proxies paradigm as OpenMASK [19], because only the referent of an object can be modified by users. However, when data is distributed on each node (homogeneous or partially replicated worlds), the users can access and modify objects locally before these changes are transmitted to other users. So it is necessary to explicitly manage the users' concurrent access to these objects to avoid inconsistencies in the virtual environment. When an object can be modified by only one user at the same time (non-collaborative interactions), Lee et al. [15] distinguish three kinds of mechanisms to manage the concurrent access:

- **pessimistic mode**, as in BrickNet [25]: This mode ensures that only one user can modify an object at the same time with a lock system. When a user wants to manipulate an object, he asks to become its owner. An object has only one owner. So if the object already has an owner, the user has to wait until this owner releases the ownership of this object. Only the current owner of an object can modify it. In this way, no concurrent access to an object can occur. However, when the network latency or the number of users are high, the necessary time to acquire the ownership of an object can be long and therefore introduces latency during interactions. BrickNet [25] uses the server to save which user is the owner of each object. This mode is difficult to set up in the case of a peer-to-peer architecture. Indeed, when a node requests the ownership of an object, it must ask all the other nodes if they own this object.
- **optimistic mode**: This mode enables users to modify objects without checking the potential concurrent access on these objects. So users can have low latency interactions with these objects. However, when a conflict occurs, it is necessary to make a correction. It can be complex and also requires that users perform their actions again.
- **prediction based mode**, as in PARADE [23] or ATLAS [15]: This mode consists in predicting for each object which users may manipulate this object to prioritize these potential owners. If the prediction is false for a user (he is not interacting with the object), it gives the ownership of the object to the next user on the list of potential owners. Generally, this prediction is based on the position and the user behavior (where they move, where they look, etc.).

In several applications, it may be useful to enable several users to manipulate a same object at the same time. Margery et al. [20] classify collaborative interactions into three categories:

- Only one user can manipulate an object at the same time. So the previous modes of concurrency control can be used.
- Several users can modify simultaneously independent parameters of a same object. So the previous modes of concurrency control can be adapted to each parameter of the objects.
- Several users can modify simultaneously co-dependent parameters of a same object. Other mechanisms must be used to combine user actions to modify appropriately the object.

4 Conclusion

We have presented a state of the art of architectures and mechanisms used in the CVE systems to maintain virtual environment consistency without disturbing user interactions. We have seen that not only the network architecture, but also the data distribution (i.e. where data is stored and executed on the network) have a strong impact on this consistency. We have presented mechanisms that enable CVE systems to achieve a time synchronization between users or to manage the concurrent access to the data of a CVE. Table 1 presents a synthesis of the solutions used in some CVE systems to efficiently maintain consistency.

This state of the art report can help to determine the most adapted system architecture in order to meet the specific requirements of a new CVE. However, this report also shows that a universal solution, which would meet the requirements of all the CVE systems, does not exist yet.

We are currently involved in the French ANR project Collaviz in which we are designing a new system architecture dedicated to collaborative scientific visualization. The project requirements impose to deal with a large amount of scientific data computed on a cluster, secured connections, standard networks, and mainstream hardware. This state of the art report may help us to find a good trade-off between the CVE consistency and the system responsiveness according to these requirements.

The large field of applications of the Collaviz project incites us to design flexible solutions. About data distribution, we expect to enable users to dynamically choose which model must be used. This choice will be made at the object level rather than at the application level, because all the objects of a virtual environment do not necessarily have the same need for consistency. About consistency maintenance, we expect to manage several groups of synchronization to enable all the users to interact in the CVE event if they have hardware limitations such as low processing power or low network capabilities.

Acknowledgements

This work was supported in part by the French Research National Agency in the ANR-08-COSI-003-01 project named Collaviz.

References

[1] C. Anthes, P. Heinzlreiter, and J. Volkert. "An adaptive network architecture for close-coupled collaboration in distributed virtual environments". In *Proc. of the ACM SIGGRAPH int. conf. on VR continuum and its applications in industry (VRCAI'04)*, pages 382–385, 2004.

[2] C. Blanchard, S. Burgess, Y. Harvill, J. Lanier, A. Lasko, M. Oberman, and M. Teitel. "Reality built for two: a virtual reality tool". In *Proc. of symp. Interactive 3D graphics (SI3D'90)*, pages 35–36, 1990.

[3] W. Broll. "DWTP - an Internet protocol for shared virtual environments". In *Proc. of the 3rd symp. on Virtual reality modeling language (VRML'98)*, pages 49–56, 1998.

[4] D. P. Brutzman, M. Zyda, K. Watsen, and M. R. Macedonia. "Virtual Reality Transfer Protocol (VRTP) Design Rationale". In *Proc. of the 6th Workshop on Enabling Technologies on Infrastructure for Collaborative Enterprises (WET-ICE '97)*, pages 179–186, 1997.

[5] J. Calvin, A. Dickens, B. Gaines, P. Metzger, D. Miller, and D. Owen. "The SIMNET virtual world architecture". In *Proc. of the IEEE Virtual Reality Annual Int. Symp. (VRAIS'93)*, pages 450–455, Sep 1993.

CVE	Network architecture	Data Distribution	Communication Protocols	Synchronization	Concurrency control
VISTEL [28]	client/server	centralized	unspecified	unspecified	none
RING [11]	client/server	homogenous replicated	UDP (*multicast*)	delayed global consistency	none (no concurrent access)
BrickNet [25]	client/server	partially replicated	UDP (*multicast*)	server synchronization	pessimistic
ShareX3D [14]	client/server	homogenous replicated	HTTP	server synchronization	pessimistic
SIMNET [5]	peer-to-peer	homogenous replicated	UDP (*broadcast*) + DIS	unspecified	none
DIVE [10]	peer-to-peer (+ a server saves the CVE state)	partially replicated	"MBone" (*multicast*)	time warp synchronization	none
OpenMASK [19]	hybrid	homogenous replicated (uses the referent/proxies paradigm for the execution of object behaviors)	TCP (with PVM for OpenMASK3 and MPI for OpenMASK4)	lockstep synchronization or delayed global consistency (possibility to specify the tolerated latency or to release the synchronization)	based on the referent/proxies paradigm (+ enables users to achieve simultaneous interactions)
CAVERN [16]	hybrid (interface enables node to be a client or a server)	partially replicated (uses a kind of referent/proxies paradigm)	TCP, UDP or *multicast*	delayed global consistency, forced from a particular node, or relaxed	based on the referent/proxies paradigm
SPLINE [26]	hybrid	partially replicated	*unicast* or *multicast* UDP, TCP, HTTP, RTP	unspecified	pessimistic
PaRADE [23]	hybrid	homogenous replicated	Unreliable *multicast* and reliable *multicast* using TCP	imposed global consistency	prediction based
Anthes et al. [1]	hybrid	homogenous replicated	*multicast* (based on a server hierarchy)	unspecified (imposed global consistency ?)	unspecified
ATLAS [15]	hybrid	partially replicated	*unicast* or *multicast*	unspecified	pessimistic, optimistic, and prediction based

Table 1: Synthesis of architectures and mechanisms used by some CVE systems for consistency maintenance

[6] J. S. Dahmann, R. M. Fujimoto, and R. M. Weatherly. "The DoD High Level Architecture". In *Proc. of the 29th conf. on Winter simulation (WSC '97)*, pages 142–149, 1997.

[7] D. Delaney, T. Ward, and S. McLoone. "On consistency and network latency in distributed interactive applications: A survey – part I". *Presence: Teleoperators and Virtual Environments*, 15(2):218–234, 2006.

[8] D. Delaney, T. Ward, and S. McLoone. "On Consistency and Network Latency in Distributed Interactive Applications: A Survey – Part II". *Presence: Teleoperators and Virtual Env.*, 15(4):465–482, 2006.

[9] C. El Zammar. *"Interactions coopératives 3D distantes en environnements virtuels : gestion des problèmes réseau"*. PhD thesis, INSA de Rennes, 2005.

[10] E. Frécon and M. Stenius. "DIVE : A scaleable network architecture for distributed virtual environments". *Distributed Systems Engineering*, 5:91–100, 1998.

[11] T. A. Funkhouser. "RING: a client-server system for multi-user virtual environments". In *Proc. of the symp. on Interactive 3D graphics (SI3D'95)*, pages 85–93, 1995.

[12] D. R. Jefferson. Virtual time. *ACM Trans. on Programming Language and Systems*, 7(3):404–425, 1985.

[13] C. Joslin, T. Di Giacomo, and N. Magnenat-Thalmann. "Collaborative virtual environments: from birth to standardization". *IEEE Communications Magazine*, 42(4):28–33, Apr 2004.

[14] S. Jourdain, J. Forest, C. Mouton, B. Nouailhas, G. Moniot, F. Kolb, S. Chabridon, M. Simatic, Z. Abid, and L. Mallet. "ShareX3D, a scientific collaborative 3D viewer over HTTP". In *Proc. of the 13th int. symp. on 3D web technology (Web3D'08)*, pages 35–41, 2008.

[15] D. Lee, M. Lim, S. Han, and K. Lee. "ATLAS: A Scalable Network Framework for Distributed Virtual Environments". *Presence: Teleoperators and Virtual Environments*, 16(2):125–156, 2007.

[16] J. Leigh, A. E. Johnson, and T. A. DeFanti. "Cavern: A Distributed Architecture for Supporting Scalable Persistence and Interoperability in Collaborative Virtual Environments". *Virtual Reality: Research, Development, and Applications*, 2(2):217–237, 1997.

[17] M. R. Macedonia and M. J. Zyda. "A taxonomy for networked virtual environments". *IEEE Multimedia*, 4(1):48–56, Jan-Mar 1997.

[18] M. R. Macedonia, M. J. Zyda, D. R. Pratt, P. T. Barham, and S. Zeswitz. "NPSNET: A network software architecture for large scale virtual environments". *Presence*, 3(4):265–287, 1994.

[19] D. Margery, B. Arnaldi, A. Chauffaut, S. Donikian, and T. Duval. "OpenMASK: Multi-Threaded or Modular Animation and Simulation Kernel or Kit : a General Introduction". In *Proc. of the Virtual Reality Int. Conf. (VRIC'02)*, pages 101–110, 2002.

[20] D. Margery, B. Arnaldi, and N. Plouzeau. "A General Framework for Cooperative Manipulation in Virtual Environments". In *Proc. of the Eurographics workshop on Virtual Environments conf. (EGVE'99)*, pages 169–178, 1999.

[21] M. Mauve, V. Hilt, C. Kuhmunch, and W. Effelsberg. "A general framework and communication protocol for the transmission of interactive media with real-time characteristics". In *Proc. of the IEEE Int. Conf. on Multimedia Computing and Systems (ICMCS'99)*, volume 2, pages 641–646, Jul 1999.

[22] D. J. Roberts, M. D. Ryan, and P. M. Sharkey. "Combining Two Techniques for Overcoming Network Delays in a Distributed Virtual Ball Game". In *the 2nd Workshop on Systems Aspects of Sharing a Virtual Reality, In proc. of the ACM conf. on Collaborative Virtual Environments (CVE 98)*, 1998.

[23] D. J. Roberts and P. M. Sharkey. "Maximising concurrency and scalability in a consistent, causal, distributed virtual reality system, whilst minimising the effect of network delays". In *Proc. of the IEEE workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 161–166, Jun 1997.

[24] C. Shaw and M. Green. "The MR Toolkit Peers Package and Experiment". In *Proc. of the IEEE Virtual Reality Annual Int. Symp. (VRAIS'93)*, pages 463–469, 1993.

[25] G. Singh, L. Serra, W. Png, A. Wong, and H. Ng. "BrickNet: sharing object behaviors on the Net". In *Proc. of the IEEE Virtual Reality Annual Int. Symp. (VRAIS'95)*, pages 19–25, Mar 1995.

[26] R. Waters, D. Anderson, J. Barrus, D. Brogan, S. Mckeown, T. Nitta, I. Sterns, and W. Yerazunis. "Diamond Park and Spline: A Social Virtual Reality System with 3D Animation, Spoken Interaction, and Runtime Modifiability". *Presence: Teleoperators and Virtual Environments*, 6(4):461–480, 1997.

[27] R. Waters, D. B. Anderson, and D. L. Schwenke. "Design of the Interactive Sharing Transfer Protocol". In *Proc. of the IEEE workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 140–147, Jun 1997.

[28] M. Yoshida, Y. A. Tijerino, S. Abe, and F. Kishino. "A virtual space teleconferencing system that supports intuitive interaction for creative and cooperative work". In *Proc. of the symp. on Interactive 3D graphics (SI3D'95)*, pages 115–122, 1995.