

# Distributed Interaction in Virtual Spaces

Alois Ferscha and James Johnson  
Department of Applied Computer Science  
University of Vienna  
ferscha@ani.univie.ac.at

## Abstract

*Virtual spaces based on the metaphor of "shared network places" are becoming a well accepted implementation approach for multiuser, multimedia, distributed cooperative work environments to support the work activities and interactions of goal oriented business teams. Similar to real-life physical team rooms, which provide a permanent shared space used by a work group, TEAMWORKPLACE, the virtual space presented in this paper, provides a virtual meeting room, i.e. a place to store, retrieve and present multimedia content (documents), to design and construct cooperatively, to brainstorm, to negotiate and make decisions, to run shared applications, to browse enterprise data, or to acquire, formalise and exchange enterprise knowledge. Since physical team rooms rely on the physical proximity of the team members and their easy access to the common work space, this access breaks down when groups are geographically dislocated. For such teams, TEAMWORKPLACE provides the electronic equivalent of a physical team room transcending distance, time zones, and organisational boundaries.*

*TEAMWORKPLACE is based essentially on two open technologies, the virtual reality modelling language (VRML) and Java. A Java-enabled Web-browser together with a VRML plug-in is the only requirement for the execution of TEAMWORKPLACE on current consumer hardware.*

**Keywords:** Virtual Spaces, Distributed Cooperative Environments, Multiuser Interaction, VRML, Java.

## 1 Introduction

With the development of VRML (Virtual Reality Modeling Language) and its establishment as an open standard for the transfer of 3D data over the Internet, the opportunity to build multiuser, distributed virtual worlds for use on a world-wide scale, independent of platform or operating system constraints, has become a reality. Owing to a worldwide dissemination of free VRML-compliant browsers, interactive 3D graphics and small scale virtual reality technologies have become readily available on any desktop. This opens a whole new range of possibilities for creating 3D content on the Internet. Together with Java, which provides the computational and networking capabilities for creating complex interactive and distributed environments, a powerful open standard based virtual reality platform is already a reality for anyone who has access to the WWW [2]. The current VRML standard (VRML97 [1]), however, does not provide support for the development of shared

multiuser worlds. A number of proposals have been made to extend the capabilities of VRML to support multiple users [10] dealing mostly with their representations as avatars and the sharing of objects and primarily based on centralized servers [11][12]. Nevertheless, does VRML97 provide the developer with some basic means necessary for the development of shared multiuser environments, i.e. the provision of two Java interfaces through which the developer may interact with the VRML world and the VRML browser itself. Thus, developers may implement the lacking multiuser and network support in the current standard by means of the Java interfaces without necessitating the definition of non-standard extensions to VRML.

### 1.1 Java / VRML based Virtual Spaces

At the current state of development of the VRML standard, research is called upon to investigate all aspects of multiuser coordination and collaboration in VRML operated distributed systems. Due to the unprecedented popularity of the TCP/IP suite of protocols and success of the WWW, the Internet holds great potential for the development of distributed multiuser applications and, especially since the introduction of Java, provides an international standards-based platform for the integration of Internet services [6][7]. However, although Java is multithreaded, it is not the ultimate language for distributed multiuser computing [8]. Sockets and RMI (Remote Method Invocation) provide methods sufficient for the development of client/server applications but complexity quickly increases when more involved forms of multiuser interaction are considered [9]. In such situations, methods for the coordination of remote Java components become more critical.

Taken together, the WWW, Java and VRML constitute a promising platform for the development of distributed virtual spaces and investigations of coordination and collaboration issues in distributed multiuser virtual environments. In this paper we present TEAMWORKPLACE, a virtual space built along these technologies. The paper is organized as follows. In Section 2 requirements for the implementation of multiuser interaction are analysed based on related work. The TEAMWORKPLACE is detailed in Section 3, where the basic interaction architecture is developed on top of the VRML event model. Section 4 presents the real-time interfacing between a VRML world and the applet that controls it on the client (browser) side. Section 5 is dedicated to showcase the use of TEAMWORKPLACE in its current stage of implementation.

## 2 Virtual Meeting Spaces: Related Work

Roughly, two groups of distributed virtual environment (DVE) research projects can be identified today. First, those dealing with large scale distributed simulation, communication issues (some using high bandwidth communication) and network

---

This work was supported by the *Oesterreichische Nationalbank*, Vienna, Austria, under grant *Jubiläumsfonds 7022*.

protocols, and second, research that concentrates on sophisticated group interaction (suitable for such work as CSCW), 3D modeling issues and management of shared resources in distributed virtual environments.

A primary consideration in the design of a multiuser DVE is the choice of the network communication structure to be realized. Basically, three possibilities exist for the communication structure of Internet-based multiuser DVEs:

- Client/Server (using TCP/IP or UDP/IP)
- (IP) Multicast (using UDP/IP only)
- Hybrid combinations of both

A good portion of DVE research work deals with protocol support for (very) large scale distributed environments which necessarily entails the use of IP multicasting. In the case of hundreds of simultaneous users, IP multicasting is certainly the only viable choice. IP multicast is an experimental Internet protocol which requires hosts to set up a dedicated connection to the multicast backbone (MBONE) which is actually a subset of the Internet defined by specialized routers. Furthermore, only UDP datagrams can be sent via IP multicasting. Despite the fact that IP multicasting provides no reliable network services, it is very efficient for sending messages to a large group of receivers due to its “pruning” mechanism. Multicast routers forward messages to other multicast routers only if at least one host in the multicast group is located in that part of the subnet. While unreliable transmission is sufficient for many multiuser applications (e.g. streaming audio/video), it is not for multiuser DVEs since each user must have a view of the environment which is consistent with that of other users so that the impression of a single, virtual environment is created. In order to compensate for the unreliability of IP multicasting, protocols for DVEs which use IP multicasting must therefore provide for mechanisms for the detection and recovery from transmission failures.

One such standard for shared DVEs in existence today is DIS (Distributed Interactive Simulation, nowadays prospered to the high level architecture HLA). DIS is an Internet protocol based on IP multicasting and is used primarily for distributed simulation of military scenarios. DIS scales up well to a large number of users and due to the limited number of entity types allowed, unreliable multicast is not a problem since hosts implement a mechanism which can detect the addition or removal of entities. Changes in the state of entities is transmitted so frequently to all participants such that lost packets can be ignored since the new state information will be received within several seconds. Furthermore, DIS uses a “dead-reckoning” technique to further optimize the simulation of the shared virtual environment. With dead-reckoning, the movement of entities is assumed to continue along a path with constant speed until state information is received to the contrary whereupon the state of the entity is gradually corrected (“track smoothing”). Due to the continuous state information transfers and dead-reckoning techniques, DIS does not appear very suitable for general purpose multiuser DVEs.

A number of protocols can be grouped under the term “reliable multicast protocols” which differ for the most part in the way in which reliability is achieved. In the domain of distributed virtual environments, several protocols are of

particular interest. MTP-2 (Multicast Transport Protocol) [5] is a multicast protocol which ensures a total ordering of packets transmitted. This ordering is achieved through the use of a central server, called the master. A member wanting to send a message (producer) must request the sending of a token from the master. The master then sends the token to the producer. The producer now tags all packets of the message with the number contained in the token and sends these to the multicast group. Consumers (i.e. those receiving packets) detect missing packets by gaps in the numbering sequence received or by a time-out value. If a packet is missing, the consumer sends a negative acknowledgement (NAK) to the master to request retransmission. The master is now confronted with the problem of dealing with (a possibly large number of) NAKs and, depending on the decision made, may choose to force retransmission of the packet or reject the message entirely, in which case the producer must reinitiate the send altogether. This approach is less suitable for DVEs since real-time interaction is much more important in virtual worlds than a total ordering of packets. Other protocols take this same token-passing/packet-ordering approach with albeit modified strategies such as RMP [14] and SLF/SRM [15].

ISTP (Interactive Sharing Transfer Protocol) [4] is a hybrid protocol with respect to the underlying protocols on which it is built: HTTP, TCP, UDP and RTP. As with DIS, multicast UDP packets are used for the update of state information. However, unlike DIS, other protocols are used for transmitting other types of data. For the transmission of scene description data (typically large files), HTTP is used. TCP is used for the reliable communication of control information and RTP is used for the transmission of streams such as audio data. The protocol is chosen on a per-object basis. With ISTP, the virtual world is subdivided into “locales” (which have a practical limit on the number of users which can be served at a time). A multicast server is dynamically chosen for each locale and each member of the multicast group has an additional connection to the server which is used for acknowledging packets and for retransmission of objects after failure detection. Similarly, DWTP (Distributed Worlds Transfer and communication Protocol) [13], also implements a hybrid protocol built on TCP and UDP (unicast and multicast). So-called daemons interact with the multicast group to detect transmission failures, recover lost packages (using unicast connections) and transmit virtual world contents to new participants.

## 2.1 Meeting Spaces

To support the cooperation of multiple users in a networked virtual environment, TEAMWORKPLACE, has been implemented - solely on standard Internet technologies such as VRML, Java and HTTP. In the design of a distributed virtual environment, the choice of network protocol is a prime consideration. When choosing the Internet protocols on which to base the implementation, the choice is basically narrowed down to a unicast (TCP or UDP) or a multicast (UDP only) mode of communication. Taking advantage of the reliable transport protocol TCP effectively limits the possibilities for the implementation to that of a client/server model while using a multicast communication protocol requires the development of a much more elaborate application level protocol in order to compensate for the unreliability of UDP.

In its software architecture, **TEAMWORKPLACE** follows a centralized server approach. Due to the security policy of most Web browsers (e.g. Netscape Navigator or Microsoft Internet Explorer), which allows applets to connect only to the server from which the applet itself was downloaded (this basically precludes all other network communication topologies like multicast and peer-to-peer communication), a client/server communication structure had to be adopted. Although these security constraints can be circumvented through the use of insecure browsers, further considerations also favor a client/server communication structure. First, the VRML world and its current state must reside (or be stored) somewhere on the Internet when no users are currently in the world. This is most conveniently achieved through the use of a dedicated server. When the server is running, the current state of the world resides in the memory of the server. When new participants join, the server simply has to send descriptions of all active objects (VRML files) and their current state to the new participant. In the case that the server must be shut down, the VRML world and its current state can be saved to the server's storage.

Of course, client/server architectures do not scale well to a large number of users since the server soon becomes the communication bottleneck due to the overwhelming amount of messages which must be forwarded. In typical cases, the number of simultaneous users in client/server DVEs has a practical limit in the range of tens of users, with loss in performance increasing as the number of users grows.

### 3 TEAMWORKPLACE – a Virtual Meeting Space

**TEAMWORKPLACE** is dedicated to support the cooperation of small groups in virtual space. Two basic requirements for sharing virtual world across the Internet are i) the transfer of the world contents to all users ii) timely distribution of all changes made to the contents (events). For the transfer of the world contents a scene description language such as VRML is required. This could typically consist of the sending of one or more large files. Alternately, especially in the case that the virtual space is partitioned into spatial regions, the transfer of the world information could be realized by the sending of the individual objects which are within the viewing horizon of the user, thus reducing the amount of the information which needs to be transferred for each individual user.

Once the files have been transmitted and a local database is established by each user, all changes occurring within the world must be transmitted to all users in order to keep the distributed scene databases consistent thus providing the fidelity of a single shared virtual world. This typically entails the propagation of "events" to all users which are relatively small in comparison to the scene description information sent when the user "logs in" to a shared virtual space.

If new objects can be introduced into the world by the users, the object's description must consequently be propagated to all users. The same applies in the case that users may provide their own avatars. In these cases, the amount of information that must be transferred is typically more than in the case of the propagation of events.

Only those objects which are within viewing distance of the participant's current position are transferred. When the participant moves in the virtual world and new objects come into

view the server then sends the descriptions of only those objects to the participant. Objects which are no longer in view can be removed from the participant's local world.

#### 3.1 Base Technologies Used in Implementation

One of the foremost goals in the development of **TEAMWORKPLACE** was to use only standard Internet technologies in its implementation, thus making such applications immediately available to anyone having access to the Internet. The structure of the application consists of the following four components:

- a Java-enabled Web browser (standard, client-side)
- a VRML97 compliant browser plug-in
- a Java applet running in the Web browser (downloaded from the Web server to the client)
- a Java server application (executing on the Web server)

##### 3.1.1 VRML Concepts

The VRML language is a general 3D scene description language with the primary construct being the node. Each scene consists of a hierarchical structure called the scene graph which is a directed acyclic graph of (nested) nodes. Most of the node types defined in the VRML specification describe 3D objects. Every node contains fields which specify its characteristics. Furthermore, nodes can be grouped together to create complex objects which can then be manipulated as a whole.

The **Shape** node is a container which contains two components: geometry and appearance. The geometry component contains exactly one node which describes the geometric characteristics of the object. The "primitive" nodes, **Box**, **Cone**, **Sphere** and **Cylinder**, are pre-defined specifications of simple polyhedra that provide quick and easy building blocks. Each contains fields to specify the dimensions of the object (e.g. radius for **Sphere**). The node **IndexedFaceSet** provides the facility for specifying the coordinates of each vertex and faces of the object, thus allowing for the construction of complex shapes.

The appearance component of the **Shape** node allows the user to specify color values and textures for the shape. The nested **Material** node permits the specification of the diffuse, emissive and reflective properties of the surface coloring and also its degree of shininess and transparency. Texture nodes allow the user to map an image (such as a GIF or a JPEG image) or even a movie (e.g. M-PEG1) to the surface of the image. The mapping of the image to the object can be controlled completely by the user and the mapping process can be imagined as if one were to apply a kind of "elastic wallpaper" to the surface of a 3D object. Thus, realistic texturing effects can be achieved.

In order to position objects, the **Transform** node, which acts as a container for other nodes, allows the author to specify a translation, rotation and scale for its tree of children nodes. The **translation** field defines the displacement in a long vector. The rotation field allows the author to specify an axis around which to rotate the object and an angle giving the amount of rotation in radians. Finally, the scale field allows the objects to be scaled by a factor along the three axes. To achieve complex transformation, multiple nested **Transform** nodes can be applied.

Further nodes, include lighting nodes (**Directional-Light**, **PointLight** and **SpotLight**), which define various types of light sources, may be placed in the scene and cannot be “seen” but are rather used to calculate the visibility, shininess and reflection of the surfaces of the objects which they illuminate. The **Sound** node allows for the inclusion of sound in the scene, the volume of which varies according to the distance of the user to the source of the sound in the scene.

At run-time, the fields of a node can be set by sending events to the node. Each event contains a new value for the field which is set immediately upon receiving the event. For example, a `set_radius` event can be sent to a **Sphere** node in order to change the current radius of the **Sphere**. The following section introduces several more nodes and explains the event model which is used in the VRML language.

### 3.1.2 The VRML Event Model

VRML uses the “point and click” method for user interaction with the scene. Several types of sensors pick up on mouse movement and clicks and produce events which can then be “routed” to other nodes. The **TouchSensor** node registers when the user clicks on any piece of its sibling geometry which is contained in the same container node as the **TouchSensor** itself. The **PlaneSensor**, **SphereSensor** and **ProximitySensor** register mouse movement and interpret it as movement in a 2D plane, on a 3D sphere or issue events when the mouse is in the proximity of the geometry, respectively.

The **TimeSensor** node detects the passage of time and emits `fraction_changed` events (which are floating point numbers between 0.0 and 1.0 inclusive) for the time it is running, i.e. from the time it is started for the length of its cycle. Thus, if the **TimeSensor** has a cycle of 10 seconds, it will emit the event {0.0} when it starts, after 5 seconds the event {0.5} (half of its cycle has elapsed) and the event {1.0} after 10 seconds.

An interpolator is often used in combination with a **TimeSensor** to create animations. The **PositionInterpolator** node accepts a floating point number between 0.0 and 1.0 and calculates a new position (an x,y,z-coordinate) based on a series of key frames. A key frame consists of a series of coordinate – floating point number pairs which define a timed trajectory through space. When the **PositionInterpolator** receives a floating point number event (called a `set_fraction` event), it interpolates the two neighboring keyValues accordingly and emits a `value_changed` event containing the interpolated value. Typically, this event is then routed to the translation field of a **Transform** container node to animate the position of the child geometry.

key	0.0	0.5	1.0
keyValue	0, 0, 0	10, 0, 0	-30, 0, 0

The above key frame series would move an object from the origin to the right along the x-axis 10 units in the first half of the animation cycle. In the second half, the object would then move left 40 units creating the impression of faster movement.

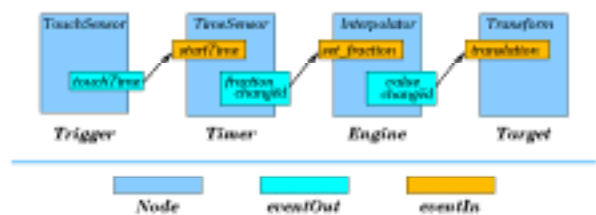


Figure1: VRML event ROUTEing

Figure 1 shows a VRML typical animation routing. In addition to fields, which remain constant during the entire existence of the VRML world, nodes which can send or receive events have corresponding `eventIn` and `eventOut` slots associated with them. When the trigger object is touched, a `touchTime` event is sent from the **TouchSensor** to the **TimeSensor** which starts the continuous generation of `fraction_changed` events in the **TimeSensor**. These are routed to the **PositionInterpolator** which calculates the interpolated position for the fraction of time elapsed. The new position is routed to the translation field of a **Transform** container node which displaces its child nodes accordingly.

An indispensable node for the developers of interactive VRML worlds is the “programmable” Script node. The Script node may have any number of user-defined `eventIns`, `eventOuts` and fields. Upon receipt of an `eventIn`, a user written function is called to handle the event. After the calculation has been performed, the user may choose to set the value of any of the `eventOuts`. Upon completion, the `eventOut` is then routed to the destination node. Through the use of Scripts, complex behaviors can be achieved.

### 3.1.3 Authoring Interfaces

In the original VRML 2.0 specification, the only mandatory language that VRML browser providers were required to implement in the Script node was JavaScript. The JavaScript dialect implemented here includes several classes needed for interfacing with the VRML world, such as types for handling VRML events. JavaScript is indeed sufficient for a vast majority of tasks of the Script node but soon it became evident that it should also be possible for developers to program the Script node with Java. In the VRML97 Standard Java is now required to be supported by all conformant browser implementations.

The desire, however, to be able to control and interact with the VRML world from outside the VRML browser was soon felt and a second Java interface was devised. The EAI (External Authoring Interface) provides an interface between the VRML world and a Java applet residing on the same page loaded in the Web browser. In order to distinguish between the EAI and the Script node interface, the former is now sometimes referred to as the JSAI (JavaScript Authoring Interface). The EAI now allows for 4 types of access into the VRML scene:

- i. access the functionality of the Browser Interface.
- ii. send events to `eventIns` of nodes in the scene.
- iii. read last value sent from `eventOuts` of nodes.
- iv. receive notification when events are sent from `eventOuts` in the scene.

The EAI thus provides all the functionality of the JSAI although being somewhat more difficult to program. In keeping with the object oriented design of Java, nodes in the VRML scene are encapsulated in Java objects which provide methods for the sending and retrieving of events.

In the context of multiuser client/server VRML applications, the EAI is the interface of choice since an applet executing at the client can provide both the networking capabilities of Java and access to the internal workings of the VRML world through the EAI.

### 3.2 TEAMWORKPLACE Implementation

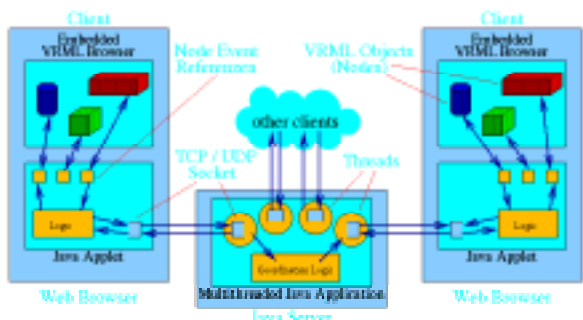


Figure 2: Architecture of the TEAMWORKPLACE Distributed Virtual Environment

The architecture of the multiuser VRML client/server application is shown in Figure 2. The HTML page for the client side of the application which is loaded into each client Web browser contains an embedded VRML browser which displays the 3D scene as well as a Java applet which acts as an intermediary between the VRML browser and the server application. The Java applet accesses events in the VRML scene by means of the EAI (External Authoring Interface). Upon initialization, the Java applet makes a socket connection to the server at a pre-defined port number. The server, which is a stand-alone Java application, spawns a new thread for each client connection which then handles all communication between the server and the client. The server then transmits current state of the shared environment to the client which displays it in the VRML browser.

When an event occurs in the VRML scene which must be echoed to other clients, the embedded Java applet receives notification of the event's occurrence by means of a callback mechanism which is provided by the EAI. The callback returns the values of the event, the (local virtual) time of its occurrence and possibly further user information associated with the event. In the typical case, the applet will send the event immediately to the server. In some cases, however, such as the rapid succession of transformation events (e.g. translation or rotation events) caused by the movement of the object in the scene, the applet's logic may decide to forward the event only when some minimum change has been exceeded, thus avoiding considerable network traffic.

The event is then received at the server by the thread which services that specific client. In most cases, the thread will then

pass the event to all other threads which in turn forward the event immediately to all respective connected clients. As with the forwarding logic built into the client-side applet, the server may choose not to forward the event to the clients (or certain clients), for example, in order to ensure consistency or fairness.

Finally, when the event is received at the clients, the receiving Java applet sends the event to the object in the VRML scene for which it was intended by means of the EAI.

#### 3.2.1 VRML Objects

In the case of a single-user VRML world, the entire description of the world (i.e. the "scene graph") is typically contained in one VRML file (with the extension ".wrl"). In the case of dynamically changing VRML worlds, however, this type of information management is not suitable since objects are typically added, removed or their attributes changed during the lifetime of the world. Furthermore, since objects can send and/or receive events, the controlling application must be able to identify objects and access them individually. This necessitates some sort of indexing or naming scheme for objects so that events which are received can be expediently routed to the correct object in the VRML scene.

For this reason, the VRML scene is considered to consist of a set of self-contained objects. Furthermore, interaction with objects is only possible through the event interface of their top-level node. In VRML, of course, any constituent node of a (compound) object (which is in fact a compound node itself) may be addressed individually through the DEF mechanism. Owing to the fact, however, that the EAI cannot access nodes by their DEF names, this type of node addressing is not possible. At first glance this would seem to be an unfortunate restriction but in fact this type of addressing is not necessary at all.

In order to deal with objects of arbitrary structure and functionality, it is necessary to standardize the way in which other objects and users can interact with objects. This can be achieved by moving all events (as well as fields and exposedFields) which should be accessible from the outside world into the top-level event interface of the object.

For instance, user interaction with the VRML representation of an electric fan will probably be limited to turning the fan on and off by clicking on its switch. When the fan is turned on, interpolation engines are started which move the blades of the fan. The constituent node of the fan object which actually initiates the turning on and off of the fan (a SENSOR node) is located at some lower level of the object's nested node structure. In a static VRML world, no top-level event interface is necessary here since the sensor node can be accessed explicitly. In a DVE, however, the fact that the fan has been turned on must be broadcast to all other participants.

To achieve this, the fan object must emit an event each time it is turned on or off. The controlling application (the applet implementing the EAI) must be able to identify from which object the event was emitted and which event this is (for the case that an object emits more than one type of event). The applet must then tag this event with an object identifier and event identifier and send it to the server for broadcast. Likewise, the fan instances in the other participating worlds must provide a means for allowing them to be turned on, "remotely", as it were. That is, the fan object must also be able to receive an event

which invokes the same effect as the touching of its sensor. This can be achieved most economically by providing corresponding events (or exposedFields) in the object's event interface.

### 3.2.2 Object Prototypes

For implementing this type of object encapsulation, VRML provides a convenient means: the prototype definition (PROTO). The prototype header consists of the word "PROTO" followed by the name of the prototype and the event interface which is enclosed in square brackets.

VRML distinguishes four types of events which may appear in the event interface of nodes and prototypes:

- i. The **field** keyword defines events which can only be received upon initialization (instantiation) of the object and cannot be subsequently changed during the life of the object.
- ii. The keyword **eventIn** defines events which the object is capable of receiving (but not sending) and
- iii. **eventOut** defines events which the object can emit (but not receive).
- iv. If the value of field can be changed at run-time, VRML provides the event type **exposedField** for defining fields which are capable of both sending and receiving events.

Each event in the event interface is defined syntactically through one of the above four keywords followed by the event type, event name and the initialization (default) value for the event.

```
PROTO Fan [
  field SFVec3f translation 0 0 0
  field SFRotation rotation 0 1 0 0
  exposedField SFBool onOff FALSE
]
{
# a compound node containing the VRML
definition of the object
}
```

The above prototype defines a possible event interface for the electric fan object. The fields translation and rotation allow for initial placement of the object in the VRML world (the default values place the object in an unrotated position at the origin of the coordinate system of the VRML world). The **exposedField onOff** is used to emit an event when the switch of the fan is touched (a Boolean value, either **TRUE** or **FALSE**). This is equivalent to an **eventOut**. Additionally, however, the **exposedField** can also receive Boolean events making it in this respect equivalent to an **eventIn**. A full prototype definition consists of the event header followed by the actual object definition (enclosed in curly braces).

Prototypes are stored by the server in the prototype database. Clients wishing to import new objects into the shared world can send new prototype definitions to the server at any time. The server, after checking for naming collisions among the objects,

then inserts (or replaces) the prototype definition in the database where it can then be referenced and sent to other connected clients.

### 3.2.3 Object Instantiations

Prototype definitions, however, do not define actual objects in the VRML world but provide only a "blueprint" for objects. In order to actually create an object and place it in the world, a prototype must be instantiated with actual values. VRML provides two means for instantiating prototypes. The first method, which is practical only in single user worlds, allows for the prototype definition to be included directly in the VRML file containing the entire scene. Instantiations of the object may be created simply by using the prototype's name as a node name (the prototype actually defines a new type of node).

The second method, which is also applicable for dynamic multiuser worlds, permits the use of object definitions which are not included in the main VRML scene file but in external files (which the VRML browser then may reference as a typical URL). These are included in the main scene as so-called **EXTERNPROTOS**.

```
EXTERNPROTO Fan [
  field SFVec3f translation
  field SFRotation rotation
  exposedField SFBool onOff
] "fan.wrl"
```

The above EXTERNPROTO definition references the PROTO definition contained in the external file "fan.wrl". As it can be seen, an EXTERNPROTO definition is similar to a PROTO definition except for the fact that the file name containing the prototype definition is used in the place of the actual compound node containing the object definition and that no default (initialization) values are provided in the event interface (these are defined in the prototype definition itself).

In order to dynamically create an instance of the fan object in the VRML world, a node of the corresponding type must be inserted into the root node of the VRML scene. The following code instantiates a node of type "Fan".

```
Fan {
  translation 10 0 0
  onOff TRUE
}
```

This places an instance of the fan object 10 units away from the origin along the positive X-axis in a unrotated position (the default rotation value in the prototype specifies no rotation). The fan is initially turned on which must be explicitly noted in the object instance since the prototype sets the **exposedField onOff** initially to **FALSE**, meaning the fan is turned off by default.

The server, besides maintaining a prototype database, also maintains a database of currently existing object instances. The file format for these objects contains an **EXTERNPROTO** definition, followed by a node instantiation. Furthermore, the information specifying which events must be propagated in the multiuser world are appended in the form of routing information

at the end of the file. An example file for the fan object is shown below:

```
EXTERNPROTO Fan [  
    field SFVec3f translation  
    field SFRotation rotation  
    exposedField SFBool onOff  
] "fan.wrl"  
  
Fan {  
    translation 10 0 0  
    onOff TRUE  
}  
  
BROADCAST this.onOff TO this.onOff
```

When processing an instantiation file, the server creates an instance of the object in its database of existing objects and assigns a unique numerical object identifier to the new object (objectID). The **BROADCAST** keyword gives information for broadcasting outgoing events in the multiuser world and is modeled after the standard **ROUTE** syntax in VRML. The reserved word "this" is used to designate current object. Numerical event identifiers (eventID) are assigned to each event sent by the object beginning with 1 (in the order of the **BROADCAST** lines). Thus, each (outgoing) event can be uniquely identified using the objectID and eventID. The server then forwards the instantiation file to all connected clients.

The clients, in their turn, pass the **EXTERNPROTO** definition and the following object instantiation "as is" to the VRML browser via the EAI. This immediately creates an instance of the object in the VRML world. The information contained in the **BROADCAST** lines is used to set up the callbacks which detect event occurrence in the VRML world and create references to the object and its events which are used for passing incoming events to the object.

### 3.3 VRML Objects and their Java Counterparts

When a VRML object is passed to the client for insertion into the local world, the EAI applet creates a Java object in its memory for representing the VRML object and the characteristics of the VRML object (objectID, the VRML instantiation string etc.) are stored in public and private variables of the Java object. Methods are provided for creating the object, setting up the callback mechanism, the sending and receiving of events and the removal of the object from the scene. Since the VRML type system recognizes a total of 20 different event types, various abstract methods in the Java object are implemented by specialized objects which are capable of handling the specified event type.

Parallel to the creation of objects the EAI maintains a list of active VRML objects in a (numbered) vector array. Thus, when it receives an event (which is tagged with the objectID), the EAI passes further control to the corresponding object for the handling of the event. Since the objectID is actually an array index, no time penalty is incurred in the searching of the database.

Similarly, the server, which is responsible for allocating objectIDs, also maintains a database of active objects. When an

event is received (on a client thread), the server inspects the objectID of the event and passes it to the corresponding object for forward to the other clients.

## 4 TeamWorkplace Showcase

As a demonstration showcase, we have constructed a virtual meeting space for workgroups on top of the TEAMWORKPLACE software architectures, following an office metaphor for environment visualization.



Figure 3: Meeting Room in TEAMWORKPLACE

Navigation within the virtual world is provided for by the VRML browser itself which is a Web browser plug-in. In Figure 3, the VRML browser is the "Cosmo Player" (<http://www.cosmosoftware.com>). The control panel provides the user with a variety of possibilities for movement within the virtual world.

The main controls can be found in the center of the control panel. The symbol at the center signals that the user is currently in "walk" mode where either mouse movement and cursor keys allows for displacement in any direction. Further controls include



Figure 4: Meeting Room: Viewpoints

“seek” mode where a click on an object automatically transports the user’s viewpoint to that object and “slide” mode which interprets mouse movement and cursor keys as a panning movement. Finally, “explore” mode allows for rotation and tilting of the entire scene. Furthermore, so-called “viewpoints” may be pre-defined by the world developer to automatically teleport the viewer to the coordinates of the viewpoint in the virtual world.

In Figure 4, a bird view has been chosen to observe the virtual meeting scene.



Figure 5: Avatars for Group Awareness

Group awareness among the participants of a meeting is (for the time being) realized using iconic avatar abstractions (Figure 5). A future version of the workspace awareness system will have live video avatars with signals coming from desktop videoconferencing systems of the participants.

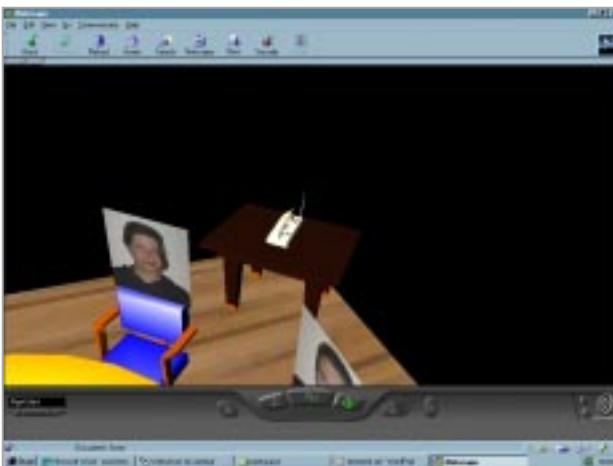


Figure 6: Audio Streaming in TEAMWORKPLACE

Voice communication is implemented using standard techniques for audio streaming with a real world metaphor (telephone) (Figure 6)

A general purpose projection screen serves to enable multimedia presentations by the respective meeting participants. In the particular example above, a standard Power Point presentation residing on the presenters home desktop computer is remotely referenced and displayed (Figure 7).



Figure 7: Presentation Screen in TEAMWORKPLACE

To guarantee for conflict free usage of the shared presentation within the scene, a light-bulb based semaphore concept for access control has been implemented (Figure 8). This solution is just one instance of the Coordination Logic component (Figure 2), and turned out to be very intuitive with respect to usability.



Figure 8: User controlled slide presentation

Links to (hyper-)documents, or connections to other VRML worlds are realized using “anchor” nodes which allow for the linking of a VRML object with a URL. When the cursor passes over an anchored object, the cursor changes shape. A click on the object opens the URL in a new browser window similar to clicking on a link in an HTML document.

Using this mechanism, its is possible to launch any local application for execution, or to display any particular document or content. In the example below, by clicking on the screen of the

“meeting archive computer”, a meeting protocol is retrieved from the owners home desktop computer, and a local word processing program is launched to display that document (Figure 9).

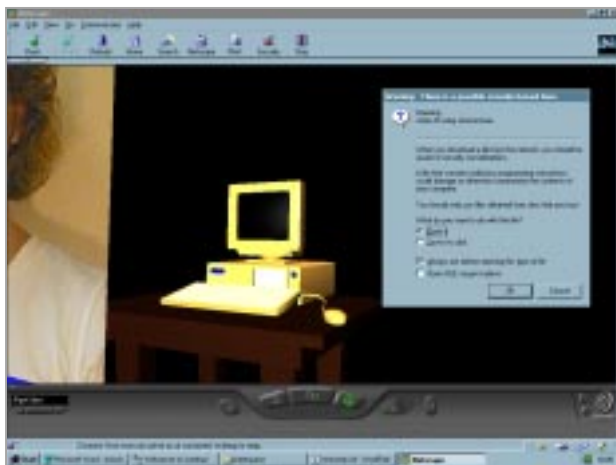


Figure 9: Launch local applications

## 5 Conclusions

Multiuser distributed cooperation environments are one of the fastest growing industrial fields of application of the Internet, particularly its most popular service, the WWW. It is becoming an indispensable vehicle for the interconnection of humans in the interactive execution of their cooperative work agenda. The software technology to support working in virtual spaces is concerned with mechanisms allowing for a group of people to connect over a network to a shared multimedia information space, to interact in real time or asynchronously on the “entities” installed in that space. This work has argued for a non-proprietary, platform and application independent cooperation software. An integrated software architecture – TEAMWORKPLACE – has been developed on top of open standards-based Web (VRML and Java). Although the current VRML standard does not provide explicit support for the development of multiuser worlds, it does, however, provide the developer with two Java programming interfaces which allow for the implementation of multiuser capabilities. Through the use of Java, which provides the computational and networking capabilities for creating complex interactive and distributed environments, and VRML, which allows for the creation of complex and interactive three-dimensional scenes. A showcase scenario of the TEAMWORKPLACE environment has demonstrated, how users can interact with each other and with common shared resources in an intuitive, three-dimensional setting.

Although TEAMWORKPLACE has demonstrated the bridging of different physical locations within a virtual space as a proof of concept, a variety of challenging questions have raised beyond software solutions. Those are e.g. related to the social protocols of interaction, the behaviour of co-manipulated objects in a shared space, the appearance of collaborators in the scene, the effective

transmission of non-verbal cues (that real-world collaborators can use effectively), the provision of communication channels that allow both public addressing as well as private conversations to occur, etc. The software technological impacts of these issue, like the real time constraints to be met and the quality of service levels to be attained, are the subject of our current investigations.

## 6 References

- [1] The VRML Consortium, [www.vrml.org](http://www.vrml.org)
- [2] G. Reitmayr, S. Carroll, A. Reitemeyer and M.G. Wagner, DeepMatrix - An Open Technology Based Virtual Environment System, ASU-CSE-TR-103098, The Visual Computer Journal, [vienna.eas.asu.edu/wagner/academic/papers/deepmatrix.html](http://vienna.eas.asu.edu/wagner/academic/papers/deepmatrix.html)
- [3] J. Locke, An Introduction to the Internet Networking Environment and SIMNET/DIS, [www.nps.navy.mil/npsnet/publications/DISIntro.ps.Z](http://www.nps.navy.mil/npsnet/publications/DISIntro.ps.Z)
- [4] R.C. Waters, D.B. Anderson, D.L. Schwenke, Design of the Interactive Sharing Protocol, Postproceedings WETICE 97, IEEE Sixth Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, MIT, June 1997, IEEE CS Press, Los Alamitos CA, [www.meitca.com/opencom/istp.html](http://www.meitca.com/opencom/istp.html)
- [5] C.Bormann, J.Ott, H.-C. Gehrcke, T.Kerschhat and N. Seifert, MTP-2: Towards Achieving the S.E.R.O. Properties for Multicast Transport, International Conference on Computer Communications and Networks 94, [www.wbs.cs.tu-berlin.de/~nilss/mtp/mtp.html](http://www.wbs.cs.tu-berlin.de/~nilss/mtp/mtp.html)
- [6] E. Evans and D. Rogers, Using Java Applets and CORBA for Multi-User Distributed Systems, In IEEE Internet Computing 1, pages 43-55, 1997.
- [7] M. Chen and J. Cowie, Java’s Role in Distributed Collaboration, Concurrency - Practice and Experience 9, 6: 509-520, 1997.
- [8] D. Lea, Concurrent Programming in Java, Addison-Wesley, 1997.
- [9] P. Ciancarini and D. Rossi, Coordinating Java Agents Over the WWW, World Wide Web, 1(2):(to appear), 1998.
- [10] Y. Honda, Mitra, B. Rockwell, B. Roehl, Living Worlds – Making VRML 2.0 Applications Interpersonal and Interoperable, Draft 2.0, 14. April 1997, [http://www.vrml.org/WorkingGroups/living-worlds/draft\\_2](http://www.vrml.org/WorkingGroups/living-worlds/draft_2)
- [11] Community Server. Blaxxun Interactive. [www.blaxxun.com](http://www.blaxxun.com)
- [12] Active Worlds Servers. Active Worlds. [www.activeworlds.com](http://www.activeworlds.com)
- [13] W. Broll, DWTP – An Internet Protocol For Shared Virtual Environments, Proceedings of the International Symposium on the Virtual Reality Modeling Language 1998, ACM, pages 49-56, 1998.
- [14] B. Whetten, T. Montgomery, and S. Kaplan, A High Performance Totally Ordered Multicast Protocol, Theory and Practice in Distributed Systems, Springer Verlag LNCS 938.
- [15] S. Jacobsen, V. Liu, C.S. McCanne, and L. Zhang, A Reliable Multicast Framework for Light-Weight Sessions and Application Level Framing, Scalable Reliable Multicast (SRM), ACM SIGCOMM 1995.
- [16] European Telework Online, [www.eto.org.uk](http://www.eto.org.uk)