TeCo3D -
A 3D Telecooperation Application based on VRML and Java

Martin Mauve[1]
Praktische Informatik IV, University of Mannheim, Germany
Siemens Telecooperation Center, Saarbrücken, Germany

ABSTRACT

In this paper we present a method for sharing collaboration-unaware VRML content, e.g. 3D models which were not specifically developed for use in a distributed environment. This functionality is an essential requirement for the inclusion of arbitrary VRML content, as generated by standard CAD or animation software, into teleconferencing sessions. We have developed a 3D TeleCooperation (TeCo3D) prototype to demonstrate the feasibility of our approach. The basic services provided by the prototype are the distribution of cooperation unaware VRML content, the sharing of user interactions, and the joint viewing of the content. In order to achieve maximum portability, the prototype was developed completely in Java. This paper presents general aspects of sharing VRML content as well as the concepts, the architecture and the services of the TeCo3D prototype. Our approach relies on existing VRML browsers as the VRML presentation and execution engines while reliable multicast is used as the means of communication to provide for scalability.

Keywords: telecooperation, VRML, Java, application sharing, multicast

1. INTRODUCTION

The increase of rendering performance on all major hardware platforms, the standardization of VRML97 [18] as an exchange format for interactive 3D objects, and the existence of the Internet multicast backbone (MBone) [12] have given rise to a solid foundation for shared 3D applications. Currently, applications which use this foundation focus almost exclusively on social events (e.g. on-line chatting and gaming) [2] or military exercises [19]. However, especially in the areas of teleteaching and telecooperation between CAD engineers, applications are needed which enable a spatially distributed group of users to share and jointly interact with standard 3D objects. We call an application which provides these services a *3D telecooperation application*.

One key aspect of a 3D telecooperation application is its ability to share *collaboration-unaware 3D models*, e.g. models which have been designed without the knowledge that they might eventually be used in a distributed environment. With this functionality existing 3D models as well as those produced using standard software, such as CAD programs and animation packages, can be utilized for telecooperation, teleteaching, and telepresentation.

TeCo3D is a joint project of the University of Mannheim and the Siemens Telecooperation Center in Saarbrücken to develop 3D telecooperation applications for collaboration-unaware VRML models. The first TeCo3D prototype, which will be presented in the following sections, supports the distribution of VRML models as well as distributed interactions and joint viewing. Key design aspects are the usage of reliable multicast for all parts of the communication subsystem, the utilization of a standard browser as the VRML execution engine, and the implementation of the prototype in Java.

In Section 2 we briefly introduce the most important aspects of the Virtual Reality Modeling Language (VRML). A general discussion of sharing VRML content is presented in Section 3. Section 4 surveys fundamental ideas and describes the architecture of the TeCo3D prototype. Based on this architecture the services provided by the prototype are discussed in Section 5. Related work is summarized in Section 6. Performance issues and items of future work are addressed in Section 7.

2. THE VIRTUAL REALITY MODELING LANGUAGE - VRML

VRML is the “file format for describing interactive 3D multimedia on the Internet” [18]. A VRML file contains the platform-independent description of 3D objects and their behavior; this description is called a ‘*world*’. In order to interact with the content of a VRML file, the user needs a *VRML browser* - just as a web browser is required to view HTML files. In fact, VRML browsers are usually plug-ins to the most common HTML browsers. During 1998, VRML has reached a large momentum,

[1] Email: mauve@pi4.informatik.uni-mannheim.de

mainly because of the release of the VRML ISO standard [18] and the inclusion of VRML browsers as standard components of all major web browsers. This makes VRML the preferred description language for importing 3D models into a 3D telecooperation tool.

We introduce VRML by examining three examples. The first one deals with the description of a static object, the second shows a simple animation, and the third demonstrates the inclusion of a programming language (Java) into a VRML model. This section can only give a limited overview of the capabilities of VRML. Detailed introductions can be found in [1] and [9]. An excellent summary of the Java-VRML interface is presented in [4].

2.1. Static world

The basic building blocks of VRML are *nodes*. Nodes contain *fields*, which can be elementary data types or nodes themselves. Usually not all fields in a node are used in a given situation. In those cases they can be omitted, with the browser supplying default values for the omitted fields. The hierarchy formed by the nodes in a VRML file is called a *scene graph*.

The example shown in Figure 1 describes a red sphere with a radius of 2 meters which has been moved 1 meter to the right and 1 meter down, relative to the point of origin. The line numbers with the colons have been included for reference only and are not part of the original VRML file.

```
01:#VRML V2.0 utf8
02:
03:Transform {
04:
05:  translation 1 -1 0
06:
07:  children [
08:    Shape {
09:      appearance Appearance {
10:        material Material {
11:          diffuseColor 1 0 0
12:        }
13:      }
14:      geometry Sphere {
15:        radius 2
16:      }
17:    }
18:  ]
19:}
```

Figure 1: VRML description of a red sphere

Line 1 of the sample code identifies this file to contain VRML version 2.0 code. The file is encoded with utf8, an international superset of ASCII. The description of the scene graph starts in line 3 with a top level `Transform` node. This node has two fields which are of interest in this example: the `translation` field, which specifies a 3D translation, and the `children` field, which is a list of the nodes affected by the translation. The only child of the `Transform` node is a `Shape` node which has two fields: `appearance` and `geometry`. While the `appearance` field describes the structure of the surface of the object (e.g. the color red) the `geometry` field identifies the object as a sphere with radius 2.

Once this file is loaded into a VRML browser, the user can navigate around the sphere, and examine the object from any point of view. However, no ‘real’ interaction with the sphere is possible. This will change in the second example.

2.2. Simple animation

Figure 2 shows the VRML code for a sphere which moves when it is clicked on. The sphere starts at the 3D position (0 10 0) then moves to (10 0 0) and concludes the animation with a movement to (0 -10 0). The total duration of the movement is five seconds.

Looking at Figure 2, one recognizes in lines 12 to 26 the description of a red sphere similar to the one in the first example. The `Transform` node has been given the name “TheTransform” by using the DEF (from DEFine) statement in line 12. The name is required in order to address events which can be produced or consumed by the node.

The Group node (lines 2 - 28) is used to form a logical union of nodes. Those nodes are contained in the children field of the Group node. In this example there are 4 children: A TouchSensor, a TimeSensor, a PositionInterpolator and the already mentioned Transform node. All of the children have been given names, so that all of them can be used as producers and consumers of events.

```
01:#VRML V2.0 utf8
02:Group {
03:  children [
04:    DEF TouchMe TouchSensor {}
05:    DEF Timer TimeSensor {
06:     cycleInterval 5
07:    }
08:    DEF TheInterpolator PositionInterpolator {
09:      key      [0, 0.5, 1]
10:      keyValue [0 10 0, 10 0 0, 0 -10 0]
11:    }
12:    DEF TheTransform Transform {
13:      translation 0 10 0
14:      children [
15:        Shape {
16:          appearance Appearance {
17:            material Material {
18:              diffuseColor 1 0 0
19:            }
20:          }
21:          geometry Sphere {
22:            radius 2
23:          }
24:        }
25:      ]
26:    }
27:  ]
28:}
29:
30:ROUTE TouchMe.touchTime TO Timer.startTime
31:ROUTE Timer.fraction_changed TO TheInterpolator.set_fraction
32:ROUTE TheInterpolator.value_changed TO TheTransform.translation
```

Figure 2: VRML description of a moving sphere

The TouchSensor is a member of the set of nodes which are provided by VRML for user interaction. Its main purpose is to generate events when it has been touched (e.g. is clicked on). These events contain the time at which the TouchSensor was touched, they are called touchTime events. An event which is produced by a node is an *event-out* of that node. In order to be of use to this example, a touchTime event-out has to be routed to another node. This is done in line 30, by means of the ROUTE statement. In this line the touchTime event-out of the TouchSensor is routed to the startTime of the TimeSensor. An event which is consumed by nodes is called an *event-in* of that node. Upon receiving the startTime event-in through the route, the TimeSensor starts ticking, generating fraction_changed events-out for the cycleInterval of five seconds (line 6). Those events contain the fraction of time (a float value between 0 and 1) of the five seconds which has passed when the event is generated. The fraction_changed events-out of the timer are routed to the set_fraction event-in of the PositionInterpolator. The PositionInterpolator contains two important fields: the key and the keyValue. Each keyValue belongs to one key; in this example the key 0 identifies the 3D position (keyValue) (0 10 0). It is the task of the interpolator to take the set_fraction event-in, compare it to the available keys and produce a corresponding output value by using linear interpolation between a pair of keyValues. If, for example, a set_fraction with the value 0.25 is received by the PositionInterpolator, it recognizes this value to be halfway between the keys 0 and 0.5 and therefore generates the output value 0.5(0 10 0) + 0.5(10 0 0). This value is sent (line 32) with the value_changed event-out of the PositionInterpolator to the transform field (line 13) of the Transform node, which results in the motion of the red sphere.

2.3. Inclusion of Java

While VRML provides all the mechanisms required for simple animations, it does not (and was never intended to) support full programming language functionality. However, this functionality is required by complex animations, which involve, for example, state information, non-linear interpolation, or network activity. In order to provide full programming language functionality, VRML provides hook-ups for existing programming languages. Those hook-ups are called `Script` nodes.

The example shown in Figure 3 makes use of a `Script` node by replacing the interpolator from the previous example with a `Script` node. This 'customized' interpolator allows the sphere to move at an increasing speed, according to the physical law of gravity. Most interesting in this example is the `Script` node in lines 13 to 17; the remaining parts are similar to those in the previous example. The `Script` node contains a reference to the code that will be executed when events are received. In this example it is the Java class GravInterp.class in bytecode format. The reference is given as an URL to provide maximum flexibility. In addition to the URL reference, the `Script` node contains the definition of the events-out and events-in which can be received or sent by the `Script` node. In this case the `Script` node receives the time as an event-in and emits the resulting position as an event-out. The computation is done by the referenced Java class, which uses the VRML-Java API for `Script` nodes to communicate with the VRML scene. The usage of `Script` nodes is not restricted to any specific programming language, even though the support of Java and ECMAScript is described in the annexes of the VRML international standard.

```
01:#VRML V2.0 utf8
02:
03:Group {
04:
05:  children [
06:
07:    DEF TouchMe TouchSensor {}
08:
09:    DEF TheTimer TimeSensor {
10:      cycleInterval 5
11:    }
12:
13:    DEF TheGravInterpolator Script {
14:      url "GravInterp.class"
15:      eventOut SFVec3f value_changed
16:      eventIn SFTime set_time
17:    }
18:
19:    DEF TheTransform Transform {
20:      translation 0 10 0
21:      children [
22:        Shape {
23:          appearance Appearance {
24:            material Material {
25:              diffuseColor 1 0 0
26:            }
27:          }
28:          geometry Sphere {
29:            radius 2
30:          }
31:        }
32:      ]
33:    }
34:  ]
35:}
36:
37:ROUTE TouchMe.touchTime TO TheTimer.startTime
38:ROUTE TheTimer.time TO TheGravInterpolator.set_time
39:ROUTE TheGravInterpolator.value_changed TO TheTransform.translation
```

Figure 3: Combining Java and VRML

In addition to `Script` nodes, VRML browsers can provide another API for programming language interaction: the External Authoring Interface (EAI) [20]. The EAI is supported by most state-of-the-art VRML browsers (such as Cosmo Player [5] and WorldView [11]) and provides methods by which a VRML browser can be controlled by external applications. Through the EAI API it is possible to load and modify VRML content as well as to observe and send events. Currently the EAI can only be accessed by Java applets running inside a web browser that contains the VRML browser as a plug-in. The specification for the EAI is under development; future enhancements, which are not supported by current browser implementations, will include an RPC-based interface, and the realization of the VRML browser as a Java AWT control.

3. SHARING VRML CONTENT

The typical approach to sharing VRML content among a spatially distributed group of users is to specifically develop the content for this purpose. The driving forces behind this approach are multi-user VRML worlds [21] which allow participants to enter a world, enabling them to interact with other participants as well as with arbitrary objects contained within the virtual world. Users are represented by 3D shapes known as avatars. Typically, the author uses specialized script nodes to provide multi-user capabilities. Using these script nodes, all parts of a multi-user world (avatars, objects and the virtual environment) are specifically designed to support a distributed usage. Therefore, we call these 3D models *collaboration-aware*.

However, the vast majority of current 3D models were not designed to be used in a multi-user VRML world. This is quite natural since multi-user worlds are only one small area where VRML can be employed. Most VRML models are generated by authoring tools, and these are unaware of potential collaboration. Further areas of application — amongst many others — include the usage of VRML as the final form for the presentation of CAD output, and in teachware models which explain complex real-world objects. We call these models *collaboration-unaware*. Special 3D telecooperation software is needed to make them available to a distributed group of users.

In order to share VRML content, a 3D telecooperation application has to fulfill two main tasks: the distribution of the VRML description, and the provision of a mechanism by which interaction with a 3D model is shared.

- **Distributing the VRML description**. This task can be accomplished by using a multi-destination file transfer to distribute the interactive VRML geometry. This mechanism sees to it that all files needed to describe a certain model are transmitted from the user who imported the model into the 3D telecooperation application to all remaining participants. While more sophisticated forms of distribution (e.g. VRML streaming) might become available in the future, they are currently no viable alternatives to file transfer.
- **Sharing user interaction**. The interaction of multiple users with a collaboration-unaware model can be realized in different ways. One possible solution would be to repeatedly send the state of the model from the floor holder to all other participants of a session. We call this *state sharing*. State sharing of 3D models has two main advantages: participants are guaranteed a consistent model, and late joins can be realized without any problems. However, there also exists a severe disadvantage: the amount of data involved in defining the state of a 3D model is very large. It can easily exceed the amount of data needed to describe the initial geometry. This makes it practically impossible to use state sharing as the only method for sharing user interaction with a 3D model.
  A different approach is to take advantage of the fact that all state changes in a VRML model are triggered by events. The sources of initial VRML events are typically user actions (such as the user clicking on an object). Those initial events trigger an *event cascade* of internal events which can change the state of the VRML model. In order to share interaction with 3D models, the application needs to transmit just the initial VRML events, relying on the VRML execution model to reproduce the same event cascade and state changes for every participant. We call this approach *input sharing*. Due to the small amount of data involved, input sharing needs only limited bandwidth. However, it makes resynchronization after failure and late joins quite difficult. The current TeCo3D prototype uses the input sharing approach while a future version will also support state sharing on-demand for resynchronization and late joins.

4. CONCEPTS AND ARCHITECTURE

The basic design idea behind the TeCo3D architecture developed in Mannheim is the perception of a 3D telecooperation application as an enhanced VRML browser. This idea is valid since a 3D telecooperation application needs the full functionality of a standard VRML browser in addition to special services for telecollaboration. In order to reuse the effort that has been put into current VRML browsers, TeCo3D relies on existing VRML browsers to supply basic VRML support. These browsers are turned into 3D telecooperation applications by augmenting them with special TeCo3D functionality.

There are two basic ways in which existing VRML browsers can be used in applications requiring VRML support. First, VRML browser source-code can be modified. This is possible because special VRML browser libraries exist [6] which have been explicitly developed for this purpose. The advantage of this approach is the flexibility gained from the source-code's availability. The main drawbacks are the fixation on a single product and the fact that currently most professional VRML browser libraries are not available free of charge.

The second way to reuse an existing VRML browser is to employ the External Authoring Interface (EAI) API. The advantages of the EAI approach are the flexibility to change VRML browsers at any time (as long as the new one supports the EAI), and the free availability of standard VRML browsers. The main disadvantage is the limited functionality of the External Authoring Interface.

Since both approaches have important advantages, the TeCo3D prototype presented here is based on the EAI approach. Another prototype based on a VRML browser library is also under development. Future tests will show whether one of them is superior in 3D telecooperation applications.

Using the EAI approach it is necessary to realize a major part of the TeCo3D prototype as an applet running inside a web browser. There are three important components of TeCo3D present within the scope of that web browser's process: The VRML browser, the TeCo3D applet and the VRML content (see Figure 4). The applet and an initial VRML model are linked to a local web page which is accessed by the user to start TeCo3D. Upon initialization the VRML browser plug-in is loaded, and the applet accesses the initial VRML model by using the EAI. This initial model contains mechanisms for getting and setting the viewpoint of the local user, and it provides a framework into which other VRML models can be inserted. The applet also displays the user interface which lets the user control the services offered by TeCo3D: Connecting to a 3D telecooperation session, importing a VRML model, and giving a guided tour through the model by coordinating the viewpoints of all participants.

In order to be able to realize input sharing, the TeCo3D applet needs a mechanism to receive events from and inject events into an imported VRML model. TeCo3D does this by requiring all sources of initial events (events which can trigger an event cascade) within the VRML model to register with the applet. Once registered, they send all initial events they generate not only to the local VRML model, but also to the TeCo3D applet which can distribute them as applicable. On the other hand, the applet is able to inject events from remote participants into a registered source of initial events, possibly triggering an event cascade within the local VRML model.

While it is conceivable to realize this enhanced functionality of initial event sources by modifying the VRML browser, this cannot be done in the EAI approach. Instead, the original VRML model is processed before being imported into TeCo3D. For a true support of sharing collaboration-unaware VRML models it is important that this processing can be done automatically, possibly at the time a VRML model is selected for presentation within TeCo3D. This guarantees that the processing is transparent to the user. As will be shown in more detail below, the processing which needs to be done for the TeCo3D prototype is simple and easily executed automatically: potential sources of initial events are replaced with pre-built, customized script nodes which have the same VRML interfaces as the original source. The remaining parts of the VRML model do not need to be changed in any way. Processing the collaboration-unaware model results in *semi-collaboration-unaware* VRML content which can be shared by TeCo3D. We call this content semi-collaboration-unaware because it was not explicitly designed to work in a cooperative environment, even though it was automatically adapted for this purpose.

In order to use the EAI while providing acceptable performance for network access, the communication subsystem was designed as a stand-alone Java application running outside of the web browser's process. The foundation of the communication subsystem is the Java library iBus [17] which offers a framework for reliable multicast. Within the iBus framework different qualities of services can be chosen, ranging from unreliable to fully reliable, and completely ordered services. Each quality of service is realized as a separate layer where either the default or a customized implementation can be used. The TeCo3D prototype uses the default iBus implementation for reliable multicast, which relies on an algorithm similar to the Reliable Adaptive Multicast Protocol (RAMP) [3] to achieve reliability on top of IP multicast.

The services of the iBus library are used by the TeCo3D communication component, which acts on behalf of the applet, exchanging events and files with peer communication components. The applet and communication component communicate via local socket connections. They exchange two types of messages: *VRML events* for the realization of input sharing as well as viewpoint synchronization, and *applet events*, which are responsible for the remaining services offered by TeCo3D

(importing a model, connecting to and disconnecting from a session). While the VRML events are routed through the communication component transparently, the applet events are interpreted by the communication component and trigger actions such as connecting to the network or initializing a file transfer.

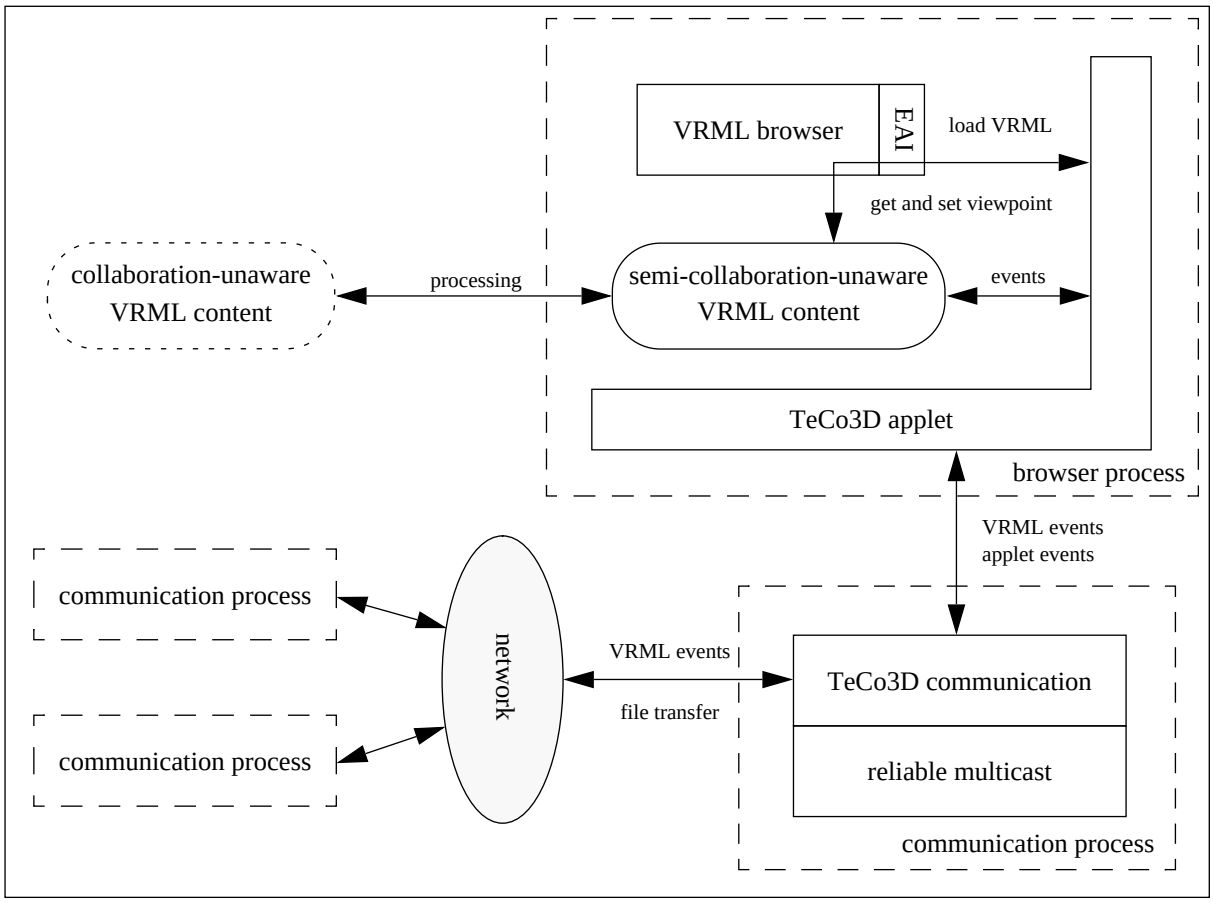

Figure 4: The TeCo3D Architecture

5. SERVICES

The current TeCo3D prototype realizes four services which turn a standard VRML browser into a 3D telecooperation application:

- connection management
- distribution of VRML content
- input sharing
- viewpoint synchronization

The following sub-sections show how these services were realized within the TeCo3D architecture.

5.1. Connection Management

In order to provide its services, TeCo3D must connect to a 3D telecooperation session. A session is identified by an IP multicast address paired with a port number. The user must provide this information along with the desired time to live (TTL) and the port where the local TeCo3D communication process is listening. As soon as this information is available, the applet connects to the communication process and requests a connection to the 3D telecooperation session by generating a connect applet event. The communication component interprets this event and asks the iBus library to establish the connection to the selected multicast address and port. After establishing this connection, iBus retrieves a list of current participants, which is

handed through the communication component to the applet, where it is displayed on the user interface. Whenever the participants of a session change, this process of setting the list of current participants is repeated. A snapshot of the user interface after the connection has been established is shown in Figure 5.

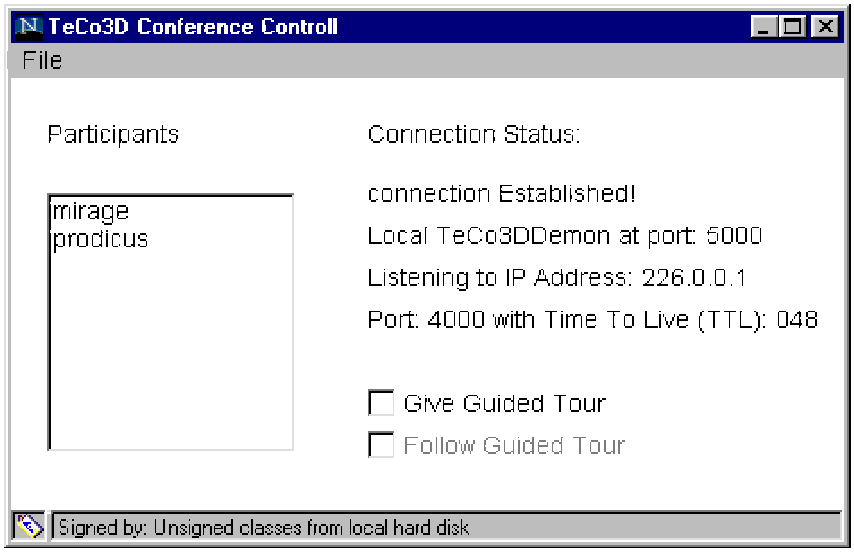

Figure 5: The TeCo3D Conference Control

5.2. Distribution of VRML Content

Once the connection to a 3D telecooperation session has been established, the user can import VRML models into the common workspace of all participants. This is done by selecting the load item from the file menu. After the user has chosen a VRML file, the applet asks the communication component to distribute the content of the directory which contains this file to all participating users. It is necessary to transfer an entire directory since a VRML model can consist of several files containing VRML geometry, texture maps, video clips, and sound. The current version of TeCo3D requires all of these components to reside in one directory on the local machine of the user. Future versions might parse and modify the VRML content so that VRML models can be taken directly from the web. The communication component is responsible for the actual file transfer. It notifies the applet when all files have been transmitted. Upon reception of this notification, the applet of each participant inserts the VRML content into the initial VRML model with an EAI call, making it visible to the user.

5.3. Input Sharing

Input sharing of VRML content is simplified by the fact that the VRML specification defines a set of standard nodes which can be utilized to realize user interaction. These nodes, which are called *sensors*, can observe user actions ranging from clicking on (touching) some part of the geometry to dragging objects and user navigation. A sensor emits an initial event when it observes a user action of the type it is listening to. The initial event can possibly trigger an event cascade, which usually leads to a state change in the VRML model. A very simple example of an event cascade was shown in Section 2, where clicking on the sphere started a timer. In general, a single user action results in one initial event which can initiate an event cascade with an arbitrary number of subsequent events.

In order to demonstrate how input sharing and the processing of VRML data works Figure 6 (a) shows an excerpt of a standard VRML file. This fragment of VRML code contains a `TouchSensor` (line 13 - 15), waiting for the user to touch (click on) some geometry. As the geometry is of no further interest in this example, it is just indicated by three dots. The second element in this example is a `TimeSensor` (lines 16 - 18), which does not react to user input but to the passage of time. Initially the `TimeSensor` is inactive. As explained in Section 2, the user can activate the `TouchSensor` by clicking on it. In Figure 6 (a) this causes the `TimeSensor` to start ticking, thereby emitting events which indicate the passage of time. These events can then be used to control an animation, like the moving sphere in the previous example.

The processing needed to transform the original VRML content into a semi-collaboration-unaware VRML model is shown in Figure 6 (b). For each sensor listening to user input, TeCo3D supplies an alternative implementation, called CooperativeX, where X is equal to the name of the original sensor. VRML provides a standardized means to integrate these customized nodes

into a VRML file by using the EXTERNPROTO statement. In this example the EXTERNPROTO statement (lines 3 - 10) is used to declare a `CooperativeTouchSensor` which is implemented in a file called CooperativeSensors.wrl. One of the key ideas behind the TeCo3D input sharing is that a customized cooperative sensor accepts the same events-in and provides the same events-out as the original sensor. Figure 6 (b) shows that the event-out `touchTime` of the `TouchSensor` is also included in the declaration of the `CooperativeTouchSensor`. All other events-in and events-out are indicated by three dots. The remaining VRML code looks very similar to that in Figure 6 (a) except that the `TouchSensor` has been changed into a `CooperativeTouchSensor` and the additional `name` attribute of the `CooperativeTouchSensor` has been added. This attribute is needed to identify that cooperative sensor which should receive an incoming event from other participants. The `name` needs to be unique within the session, and can be of the form "UserName:ProcessID: IPAddress:NodeNumber".

```
 1 #VRML V2.0 utf8
 2
 3
 4
 5
 6
 7
 8
 9
10
11 Group {
12   children [
13     DEF TouchIt TouchSensor {
14
15     }
16     DEF Timer TimeSensor {
17       cycleInterval 5
18     }
19     ...
20   ]
21 }
22 ROUTE TouchIt.touchTime TO Timer.startTime
```

(a)

```
#VRML V2.0 utf8

EXTERNPROTO CooperativeTouchSensor [
  field        SFString name
  eventOut     SFTime   touchTime
  ...
]
[
 "../CooperativeSensors.wrl#CoopTouchSensor"
]
Group {
  children [
    DEF TouchIt CooperativeTouchSensor {
      name "mauve:13089:134.155.48.116:1"
    }
    DEF Timer TimeSensor {
      cycleInterval 5
    }
    ...
  ]
}
ROUTE TouchIt.touchTime TO Timer.startTime
```

(b)

Figure 6: Examples of VRML Content Processing

It is important to notice that the transformation of the VRML model can be done automatically without human intervention. This ensures that the user perceives TeCo3D to provide true support for sharing collaboration-unaware VRML models.

Figure 7 is a code excerpt of the CooperativeSensors.wrl file which shows how the `CooperativeTouchSensor` is implemented. The sensor contains two parts: a regular `TouchSensor` and a `Script` node referencing a Java class. An object of the referenced class is generated whenever a `CooperativeTouchSensor` is used in an imported VRML file. This object registers with the applet as soon as it is created, establishing the link between VRML content and TeCo3D applet.

The `TouchSensor` (lines 9 - 12) is responsible to provide the functionality of a standard `TouchSensor` for the `CooperativeTouchSensor`. The output of the `TouchSensor` is routed (line 21) to the Java object of the `Script` node. Whenever an event is generated by the `TouchSensor`, this Java object notifies the applet and makes the event available as an output for the `CooperativeTouchSensor` (line 18). The applet uses the communication component to distribute the event to peer instances of TeCo3D. When the event is received by the remote applet it is sent directly to the appropriate Java object of the remote `CooperativeTouchSensor`. The remote `CooperativeTouchSensor` then produces an event-out as if it came from the local TeCo3D user.

Notice that sensors are not the only sources of initial events. Another VRML element which can potentially generate initial events is a customized script nodes. While the current version of the TeCo3D prototype does not support script nodes generating initial events, work is in progress to enable this in the next version.

```
 1 #VRML V2.0 utf8
 2
 3 PROTO CoopTouchSensor [
 4   field        SFString name “”
 5   eventOut     SFTime   touchTime
 6   ...
 7 ]
 8 {
 9   DEF TheTouchSensor TouchSensor {
11     ...
12   }
13   DEF TheCooperativeSensor Script {
14     url      “TeCo3D.netsensors.CooperativeTouchSensor.class”
15     field    SFString  theName IS name
16     ...
17     eventIn  SFTime    set_touchTime
18     eventOut SFTime    touchTime_changed IS touchTime
19   }
20   ...
21   ROUTE TheTouchSensor.touchTime TO TheCooperativeSsor.set_touchTime
22 }
```

Figure 7: Implementation of Cooperative Sensors

5.4. Viewpoint Synchronization

Viewpoint synchronization is a mechanism by which one participant (the guide) in a 3D telecooperation session can give a guided tour through shared 3D models. While viewpoint synchronization is in effect, the viewpoints of the guided users follow the guide, allowing all participants to focus on the same part of a model.

TeCo3D realizes the mechanism for viewpoint synchronization by including two VRML nodes into the initial VRML model. One of those nodes is a `ProximitySensor` which reads the position and orientation of the local user. The other node is a `Viewpoint,` which can be used to set the position and orientation of the local user. The functionality of these two nodes is accessed by the TeCo3D applet with EAI calls. The applet is notified when the viewpoint of the guide changes and, in response, sends a VRML event to the communication component. This event contains the orientation and position of the guide and is distributed by the communication component to all peer TeCo3D instances. As soon as the applet of a guided participant receives the information, it sets the new values of the `Viewpoint` node, changing the user’s point of view and orientation.

In order to use the viewpoint synchronization mechanism, TeCo3D implements a simple policy. Users can decide at any time whether or not they want to participate in guided tours. They can do this by using the checkbox “Follow Guided Tour” on the TeCo3D user interface (Figure 5). If they decide not to follow guided tours, their viewpoint is not changed by viewpoint synchronization. A user can choose to become the guide by checking “Give Guided Tour” during a session. The viewpoint synchronization mechanism is then initialized with all users who have chosen “Follow Guided Tour” to be true. An already existing guide is reduced to the regular user status whenever someone else becomes guide.

6. RELATED WORK

An obvious approach to sharing cooperation-unaware VRML content is the usage of general application-sharing tools. Application sharing is realized by intercepting the communication between applications and the window system. By doing this, the application-sharing component is able to multiplex the output of an unmodified single-user application to multiple users and to forward their actions to the actual application. For an example of application sharing see [14] and [13].

While sharing the output of a single VRML browser instance might seem to be similar to the functionality offered by the TeCo3D prototype, the application-sharing approach has severe disadvantages. First, and most important, the volume of data produced by sharing the output of a VRML browser is extremely high compared to sharing the output of regular applications (e.g. word processing software) whose window content does not change rapidly. At a refresh rate of 15 to 25 frames per second, the transmission of 3D-rendered window content becomes infeasible. A second disadvantage of the application-sharing approach is its limited functionality for the support of collaborative work, e.g. application sharing would not allow to mark the location of the participants in the virtual world, or enable participants to have different viewpoints.

A prototype of a shared VRML viewer was developed in the context of the Java-Enabled TeleCollaboration System (JETS) at the University of Ottawa [16]. JETS is a programming interface and toolbox for the development of collaborative Java applets. With JETS, a developer can use the mechanisms provided for consistency, access control and data exchange to develop so-called shared applets. Due to the Java security constraints, the JETS architecture is client-server oriented with all the well-known disadvantages of a centralized architecture. The shared VRML browser — a sample applet constructed with JETS — allows for the shared viewing of static VRML 1.0 content. In contrast to the JETS sample applet, the TeCo3D prototype is highly scalable because of the replicated architecture and the usage of multicast. Moreover, we allow the synchronized interaction with VRML content, which is not possible in the JETS VRML applet.

7. STATUS, LESSONS LEARNED AND FUTURE WORK

The implementation of the first TeCo3D prototype has been completed. All service as described above have been realized. The prototype proves that a 3D telecooperation application can be built on the foundation of existing VRML browsers, reliable multicast, and Java. Both performance and functionality have been shown to be acceptable. However, there are also several areas where the prototype can be significantly enhanced by future work:

- *Performance and architecture*: the current separation of the browser from the communication process places unnecessary strain on the overall performance of TeCo3D. Since VRML browsers will be available as AWT components or as a Java 3D implementation in the near future, a monolithic architecture with one process, one Java Virtual Machine, and no web browser will soon be feasible.
- *Input sharing*: the current prototype does not take into account script nodes as sources of initial events. This will be changed by providing special wrappers for script nodes which will be applied when the original VRML content is processed.
- *State sharing*: in order to realize late joins, a mechanism for transparently getting and setting the state of a VRML model will be developed and integrated into TeCo3D.
- *Reliable multicast*: the currently used mechanism for reliable multicast is sufficient for networks with a low packet loss rate. However, when packet loss is more frequent, this mechanism becomes less favourable. Future versions of TeCo3D will have a mechanism for reliable multicast similar to SMP [7], which is tolerant to higher packet loss rates. In addition we will experiment with using forward error correction and state resynchronization instead of reliable multicast.
- *Application layer protocol*: the current prototype uses a very simple and proprietary application layer protocol to distribute VRML data and events. Since a 3D telecooperation application needs real-time communication, the usage of the real-time transport protocol (RTP) [15] is appropriate. RTP not only supplies valuable timing information, but also provides means to monitor the quality of service and to convey information about the participants of a session. In order to use RTP within the TeCo3D project, we will define an RTP payload type for VRML data and events.
- *Enhanced functionality*: while the current services provide basic support for 3D telecooperation, there exists a multitude of functionality which can further enhance this cooperation. Examples are annotations of VRML content, flexible floor control [10], secure cooperation [8], as well as recording and playback of telecooperation sessions.

8. CONCLUSION

This paper introduced the concept of sharing *collaboration-unaware 3D content* as a new form of telecooperation which can significantly enhance the areas of teleteaching and telecollaboration among CAD users. The term *3D telecooperation application* has been proposed to describe applications which realize this concept.

A prototype of a 3D telecooperation application was developed within the TeCo3D project, demonstrating that the idea of sharing collaboration-unaware VRML content is valid. The prototype was developed completely in Java, using reliable multicast for the communication with peer instances. We have shown that this prototype can be realized by using an architecture which utilizes existing VRML browsers as 3D presentation and execution engines.

A set of basic services has been identified which is supported by the TeCo3D prototype: connection management, distribution of VRML content, joint interactions, and viewpoint synchronization. An emphasis was put on the presentation of a mechanism called *input sharing*, which is needed to enable joint interactions.

First results show that the performance and the services provided are surprisingly good for an application which relies heavily on Java for program control and network activity. A number of items for future work have been found, ranging from an improved architecture to additional services. The realization of these items is currently under way and should result in a fully functional 3D telecooperation application.

ACKNOWLEDGEMENTS

The TeCo3D project is sponsored by the Siemens Telecooperation Center (STZ) in Saarbrücken. Several important requirements and areas of application have been identified in discussions with the STZ research staff and its leader Dr. Jean Schweitzer. Without their support the TeCo3D project would not have been possible.

REFERENCES

[1] Ames, A. L.; Nadeau, D. R.; Moreland, J. L.: *VRML 2.0 Sourcebook*, Second edition, John Wiley & Sons, New York, 1997.

[2] blaxxun: *Colony City*, 1998. On-line: http://www.colonycity.com/index.html

[3] Braudes, R.; Zabele, S.: *Requirements for Multicast Protocols*, Network Working Group, Request for Comments 1458, May 1993. On-line: ftp://src.doc.ic.ac.uk/rfc/rfc1458.txt

[4] Brutzman, D. P.: "The Virtual Reality Modeling Language and Java", Communications of the ACM, Volume 41, Number 6, pp. 57-64, June 1998.

[5] Cosmo Software: *Cosmo Player 2.1 Features*, June 1998. On-line: http://www.cosmosoftware.com/products/player/features.html

[6] Diefenbach, P.; Mahesch P., Hunt D.: "Building OpenWorlds (TM)", *Proceedings of VRML'98*, pp. 33-38, February 1998.

[7] Grumann, M.: *SMP - A Scalable Reliable Multicast Protocol*, in German, Master's Thesis at the University of Mannheim, Lehrstuhl Praktische Informatik IV, February 1997. On-line: http://www.informatik.uni-mannheim.de/informatik/pi4/projects/teleTeaching/publikationen.eng.html

[8] Geyer W.; Weis, R.: "A Secure, Accountable, and Collaborative Whiteboard", *Proceedings of IDMS'98*, Plagemann, T., Goebel; V. (Eds.), Vol. 1483 of LNCS, pp. 3-14, Springer, Berlin, September 1998.

[9] Hartman, J.; Wernecke J.: *The VRML 2.0 Handbook*, Addison-Wesley, Reading Massachusetts, 1996.

[10] Hilt, V.; Geyer, W.: "A Model for Collaborative Services in Distributed Learning Environments", *Proceedings of IDMS'97*, Steinmetz, R.; Wolf, L. (Eds.), Vol. 1309 of LNCS, pp. 364-375, September 1997.

[11] Intervista: *WorldView 2.1 - VRML 2.0 Plugin - Developer's Guide*, February 1998. On-line: http://www.intervista.com/worldview/2-1-docs/dev-guide.html

[12] Macedonia, M. R.; Brutzman, D. P.: "MBone Provides Audio and Video Across the Internet", IEEE Computer Magazine, Volume 27, Number 4, pp. 30-36, April 1994. On-line: http://www.mbone.com/mbone/references.html

[13] Mauve, M.: *Protocol Enhancement and Compression for X-Based Application Sharing*, Master's Thesis, Lehrstuhl Praktische Informatik IV, University of Mannheim, 1997.

[14] Minenko, W.: *Advanced Design of Efficient Application Sharing Systems under X Window*, Ph.D. Thesis, Department of Computer Science, University of Ulm, January 1996.

[15] Schulzrinne, H.; Casner, S.; Frederick, R.; Jacobson, V.: *RTP: A Transport Protocol for Real-Time Applications*, Internet Draft, Internet Engineering Task Force, Audio-Video Transport Working Group, August 1998.

[16] Shirmohammadi, S., Georganas, N. D.: "JETS: a Java–Enabled TeleCollaboration System", *Proceedings of IEEE ICMCS*'97, Werner, B. (Ed.), pp. 541-547, The Printing House, June 1997.

[17] SoftWired: *iBus*, 1998. On-line: http://www.softwired.ch/ibus/

[18] VRML Consortium: *Information technology -- Computer graphics and image processing -- The Virtual Reality Modeling Language (VRML) -- Part 1: Functional specification and UTF-8 encoding*, ISO/IEC 14772-1:1997 International Standard, December 1997. On-line: http://www.vrml.org/Specifications/

[19] VRML Consortium - DIS-Java-VRML Working Group: *Project Overview*, May 1998. On-line: http://www.stl.nps.navy.mil/dis-java-vrml/

[20] VRML Consortium - External Authoring Interface Working Group: *Proposal for a VRML97 Spec Addition - External Authoring Interface Reference*, August 1998. On-line: http://www.vrml.org/WorkingGroups/vrml-eai/proposals/justin/proposal.html

[21] VRML Consortium - Living Worlds Working Group: *Living Worlds - Making VRML 2.0 Applications Interpersonal and Interoperable - Draft 2*, April 1997. On-line: http://www.vrml.org/WorkingGroups/living-worlds/draft_2/index.htm