OCT 20 2000

SANDIA REPORT

SAND98-2664
Unlimited Release
Printed December 1998
Second Printing October 2000

Multimedia Feedback Systems for Engineering

Michael J. McDonald, Eric J. Gottlieb, Scott Gladwell and Cara L. Slutter

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.

Approved for public release; further dissemination unlimited.

Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865)576-8401
Facsimile: (865)576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.doe.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800)553-6847
Facsimile: (703)605-6900
E-Mail: orders@ntis.fedworld.gov
Online order: http://www.ntis.gov/ordering.htm

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

RECEIVED
NOV 02 2000
OSTI

SAND98-2664
Unlimited Release
Printed December 1998
Second Printing October 2000

The only change to this report is the distribution limitation, which has changed from Internal Distribution Only, Patent Caution and Sandia Commercially Valuable Information to Unlimited Release

Multimedia Feedback Systems for Engineering

Michael J. McDonald, Eric J. Gottlieb,
Scott Gladwell and Cara L. Slutter
Intelligent Systems and Robotics Center
Sandia National Laboratories
P.O. Box 5800-1004
Albuquerque, NM 87185-1004

Abstract

The world wide web has become a key tool for information sharing. Engineers and scientists are finding that the web is especially suited to publishing the graphical, multi-layered information that is typical of their work. Web pages are easier to distribute than hardcopy. Web movies have become more accessible, in many offices, than videos. Good VRML viewing software, bundled with most new PCs, has sufficient power to support many engineering needs.

In addition to publishing information, science and engineering has an important tradition of peer and customer review. Reports, drawings and graphs are typically printed, distributed, reviewed, marked up, and returned to the author. Adding review comments to paper is easy. When, however, the information is in electronic form, this ease for review goes away. It's hard to write on videos. It's even harder to write comments on animated 3D models. These feedback limitations reduce the value of the information overall.

Fortunately, the web can also be a useful tool for collecting peer and customer review information. When properly formed, web reports, movies, and VRML animations can be readily linked to review notes. This paper describes three multimedia feedback systems that Sandia National Laboratories has developed to tap that potential. Each system allows people to make context-sensitive comments about specific web content and electronically ties the comments back to the web content being referenced. The first system ties comments to specific web pages, the second system ties the comments to specific frames of digital movies, and the third ties the comments to specific times and viewpoints within 3D animations. In addition to the technologies, this paper describes how they are being used to support intelligent machine systems design at Sandia.

Contents

Figures

1 Introduction

The purpose of Computer Aided Design (CAD) and Computer Aided Engineering (CAE -- including simulation) tools are to help the engineer develop and refine design concepts and generate information that others can use to produce the designs or further the concept. As a result, the tools are often built to be result-oriented. That is, the tools are typically designed to allow the engineer to develop an idea and pass the result on to others.

Conversely, in collaborative, multi-person efforts, the need is to allow several people to review the work while it is still in process. This means that the data must be accessible at various stages of completion. For example, CAE models must be reviewed before they are ready to produce answers or machine tool programs.

Traditionally, engineers who wish to review models must use the software used to create the data. Unfortunately, as the Computer Aided Design Report[1] (CAD Report) points out

> Modern manufacturing organizations employ many types of CAD, graphics and text files in hundreds of different formats. People—including planners, purchasing agents, production workers and component suppliers – need to view and use these files ... CAD and engineering software is too complex and too costly to put on every worker's desk just to view design data.

Several commercial software packages have been developed to allow people to view various forms of CAD data. These basically fall under the categories of general viewing software and data-specific codes. For example, Adobe Exchange and Acrobat are often used to generate and share electronic paper copies of CAD drawings (as well as to share electronic paper versions of output from other programs.) The CAD Report provides a good description of the state of the art in this domain:

> There are dozens, if not hundreds, of document and model viewers available. Some proprietary viewers will interpret only one type of file. Most viewing software used in conjunction with engineering systems must handle many different formats.... Unfortunately, no one program handles all available formats.... If your engineering drawings are fully dimensioned and workers are not supposed to scale them, then the best formats are compressed raster or Adobe PDF. Both have been used successfully by a number of companies. Both formats are difficult to alter. Both can be marked up using various viewing and markup programs. Both PDF and raster are derivatives of printer- and plotter-output formats, so most CAD systems can produce them.
>
> ...
>
> Some proprietary viewers allow workers to measure entities on CAD drawings. Native CAD viewers also have the advantage of not requiring conversion or processing before distribution. And recipients with the right software can modify them. The disadvantages of native CAD formats are several. First, CAD files can be altered by anyone with the right software. Second, CAD files are hard to read accurately.... (Third) is that few companies use one CAD system anymore. Supporting a multiplicity of file formats adds to the cost of electronic document distribution while providing few benefits to users of the data.
>
> ...
>
> Some makers of viewing software are now trying to provide ways to view 3D CAD data, but accurate presentation of 3D data is an order of magnitude more difficult than 2D viewing.... It also is difficult to associate comments and edits with 3D geometry. AutoVue ... and Myriat 4.0 ..., for example, can display 3D data, but comments and notations are done on 2D views of the data. It may take a large number of these 3D snapshots to document needed changes on a model.... IntraVision 3.0 lets users attach a comment to a 3D model, rotate the part, and attach another comment to another view. (SolidWorks gives its viewer away. Parametric Technology charges nearly $1,000 for its viewer).

For 3D animated simulations, such as those produced by IGRIP, the situation is even worse. Currently, the only commercially available Deneb's IGRIP simulation viewing software is, for example, the original $70K simulation software configured with the same options as the original, or a run-only version costing $25K plus options. Marketing of a low-cost ($3K) "simulation viewer" is being considered. Lack of cross-platform support further limits the value of these viewing packages.

For most 3D simulators, the alternative is to create a movie (whether in tape or its digital equivalent). While valuable, movies lack much of the information of the original and are, in addition, difficult to share. Beyond the obvious loss of 3D-scene navigation, video movies, for example, have the same resolution as the smallest available computer monitors and, for practical reasons, digital movies must often be generated at even lower resolutions. For example, a good 30-second video-resolution (640 x 480 pixel, 30 fps) digital movie can take several hours of computer processing to generate (compress) and the resulting movie will typically be several to several hundred megabytes in size. As a result, most movies are generated at ½ or ¼ frame resolution and lower frame rates.

Further limiting the practicality of sharing 3D data is the related problem of generating comments in the 3D outputs. As noted, by CAD Review,

> Managers and engineers who review drawings often have to make comments or edits on drawings or other documents.... After reviewing the 3D viewing software of a number of leading companies, we're forced to conclude that this software is not ready for mass distribution.... We found the user interfaces to be awkward and the on-line help to be baffling.... The lighting models employed by most of the 3D viewers we looked at are grotesque.... The measuring tools leave much to be desired. Most don't handle assemblies well. Most firms will not yet find 3D model distribution to be a substitute for drawings.

Finally, engineered systems are not only about CAD and CAE models. Typically, they include descriptive text, tabular data, pictures, sketches, and process diagrams. This combined information needs to be collected in context and reviewed as collections, and not as individual elements. Currently, the HTML and PDF formats are among the few electronic alternatives for producing this information as collections. Here, HTML has the advantage of providing efficient (smaller files, less data transfer) ways for presenting the collections[2] while the PDF format has the advantage of presenting the material in easy-to-print representations. Fortunately, the openness of both PDF and HTML support easy development of hybrid complex documents.

As noted earlier, the PDF format does let reviewers make comments on specific content. The mechanisms are very easy to use. However, because the comments are stored within the original file, it is difficult to collect comments from several people. Here, an editor must collect the individual PDF files and then reproduce the comments in a single "master." In addition, electronically transferring larger files (i.e., files with movies) just to transfer a few bytes of comment text can be cumbersome.

The needs, therefore, are for electronic formats that can serve a broad variety of engineering purposes and be adapted to electronic feedback systems that work as a natural extension to those forms. The remainder of this document describes three technologies that we developed to satisfy these needs. Section 2 describes an electronic feedback system that interoperates with regular web pages to allow reviewers to comment on web page content. Sections 3 and 4 describe a feedback tool that allows engineers to add annotated bookmarks to normal digital movies and animated 3D VRML models. In addition, Section 4 describes unique VRML construction techniques that allow models to be easily reviewed. Sections 5 and 6 discuss applications where these tools have been tested and provide closing remarks about the technologies.

2 Web Page Commenting

Sandia's web page commenting feedback system first appears to reviewers as a "feedback" link on each important web page of a report (see Figure 1). This link has a similar function as typical "mailto" links. Here, however, clicking on this link brings the user to a highly automated "make a comment" web form where new comments can be added or other people's comments can be reviewed. In addition to making comments, users can view other's comments through a linked comment query page.

Figure 2 shows the basic process of submitting and viewing comments. When opened from a Source Page, basic information is automatically filled into the comment form on the Make Comment Page. This information includes the URL of the page the reviewer was viewing when they clicked the feedback button, the date, and the reviewer's name and email address (if the user has made comments in the past). The reviewer then writes and categorizes the comment, makes any necessary changes (e.g., to add or change their name or email address), and submits the information to the web server.

Later, users submit search terms through a web form on the Report Query Page to generate custom reports of reviewer comments (Report Pages). These reports display the comments and provide hyperlinks to the Source Page that the comment refers to. Additional links let the reviewer respond directly to any prior comment or send email to the reviewer. Comments about comments are entered as above. When reviewed, these comments provide hyperlinks that generate custom reports that show the originating comment (i.e., links to generate pseudo Source Pages), rather than the Source Page being commented on.

2.1 Passing URLs From Source To Comment Page

Specialized JavaScript code is used on the Source and Make Comment Pages of the commenting system to automatically fill in commenting data. A simple JavaScript call is used on the Source Page to attach its web page location (URL) to the URL for the Make Comment Page. A set of JavaScript functions are used on the Make Comment Page to extract the Source Page URL, recall user information (from cookies) and copy this data into the form fields.

The Make Comment Page URL accepts data in the same format as is generated by a "GET" form submission that sets a variable called *came from*. For example, if the Source Page and Make Comment Page were located at http://web_server/link/... and http://database_server/../add_comment.htm respectively, then calling the Make

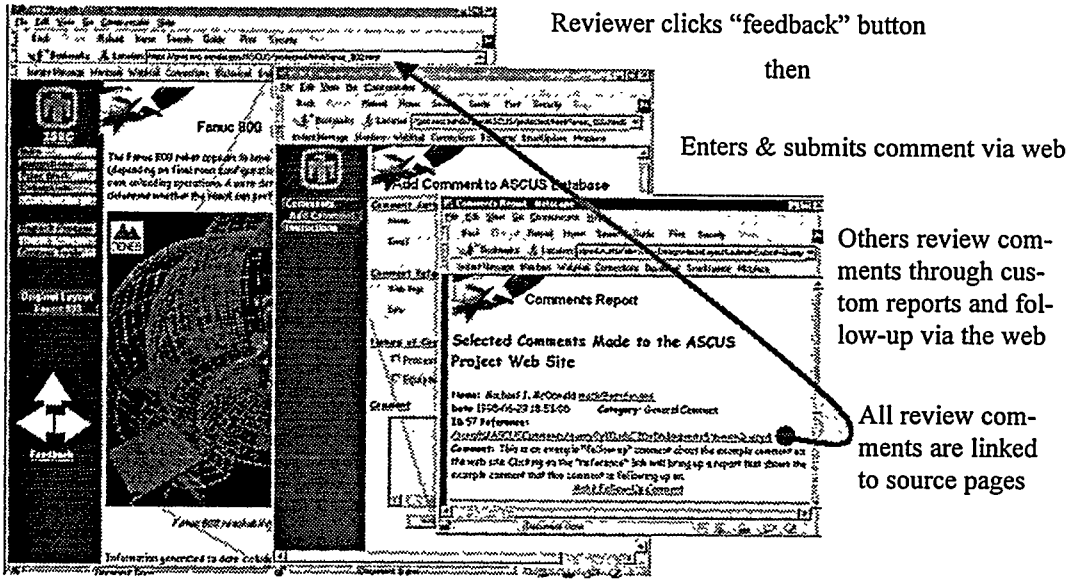

Figure 1: Order of pages user sees when making a comment via the web.

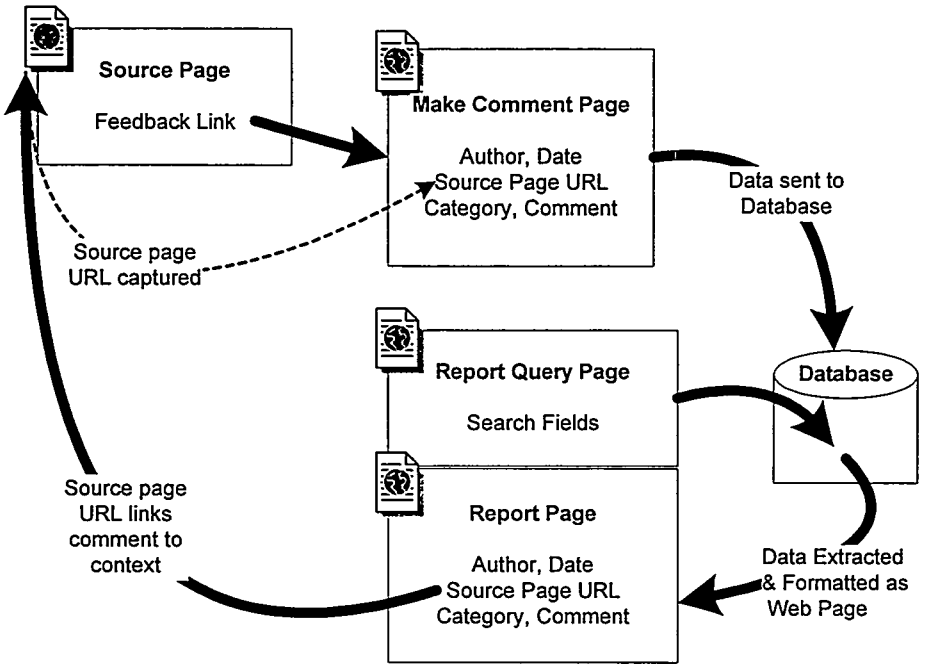

Figure 2: Basic elements of commenting system

Comment Page with the following call would provide it with the Source Page location:

```
http://database_server/../add_comment.htm?came from=http://web_server/link/... &
```

Because this is a normal web call, a specific link could be generated for each web page of a report. Unfortunately, this solution would be very difficult to maintain. To reduce the need to generate custom HTML code for each page, the "feedback" link can be automatically generated using a JavaScript call. This call can then be wrapped inside an additional call to provide minimal functionality for users of browsers that don't have JavaScript. The resulting script is as follows:

```
<A HREF = http://database_server/…/add_comment.htm>
<SCRIPT language ="JavaScript"> <!-
document.write("</A><A HREF = http://database_server/…/add_comment.htm?came from="
              + location.href + "&>")
//-></SCRIPT>>
feedback </A>
```

Within the Make Comment Pages, a JavaScript function called WhereWasI() is used to extract and parse the Source Page URL. This function is called from a JavaScript page and executed after the page is built. WhereWasI()is reproduced below.

```
// Reads the search string to figure out what link brought it here
function WhereWasI() {
  // Store search string in local variable
  var handyString = window.location.search;
  // Find the beginning of the URL variable
  var startOfSource = handyString.indexOf("came from=");
  // If the variable is defined, find the end of it
  if (startOfSource != -1) {
    var endOfSource = handyString.indexOf("&", startOfSource+9);
    // Look for special database query terminator
    var end2 = 0;
    end2 =  handyString.indexOf("!", startOfSource+9);
    if (endOfSource > end2)
```

```
        var result = handyString.substring(startOfSource+9,
                     endOfSource);
    else
         var result = handyString.substring(startOfSource+9, end2);
  }
  else
    var result = "Unknown"; // Could not find the "came from" string
  return result;
}
```

The WhereWasI() script is invoked, for example, within the body text to fill the FORM named MyForm's data INPUT named Reference with the following code:

```
<SCRIPT language ="JavaScript"> <!-
       document.MyForm.Reference.value = WhereWasI()
//-></SCRIPT>>
```

2.2 *Remembering User Data and Other Tricks*

Additional JavaScript code is used to reuse user information (name and email addresses) for successive comments. The web browsing software's "cookie" mechanisms are used to store and retrieve this user information. Custom date and formatting functions are used to correctly format the date to conform with database requirements. Form filling calls, like those used to fill URL data above, is used to write user data and dates onto the form for user editing. This section describes how these functions are used.

Cookies are data packets that are stored on the user's computer and are associated with web pages, collections of pages, or whole domains. A JavaScript can create a cookie by setting the object document.cookie to a string that contains the name and value of the cookie. An escape function is provided for encoding special characters including spaces. Additional data, including how long the computer should store the cookie (after which, it's deleted) or whether pages served from the current or other computers can look at the cookie can be added to the cookie string.

For example, the following line of JavaScript would set a cookie value with an expiration date and access qualifiers.

```
document.cookie = "foo=" + escape(value) + expString + pathString + domainStrin
```

The following generalized JavaScript functions are used to store and extract user information. An examination of the code shows that the User Name and Email address values are made available to all isrc.sandia.gov domain computers and that these values expire three months after a comment was made.

```
// SetCookie - Adds or replaces a cookie. Use null for parameters
//             that you don't care about
function SetCookie(name, value, expires, path, domain, secure) {
  var expString = ((expires == null)
                   ? "" : ("; expires=" + expires.toGMTString()))
  var pathString = ((path == null) ? "" : ("; path=" + path))
  var domainString = ((domain == null)
                   ? "" : ("; domain=" + domain))
  var secureString = ((secure == true) ? "; secure" : "")
  document.cookie = name + "=" + escape(value)
                  + expString + pathString + domainString
                  + secureString;
}

function saveUserID(){
       var ThreeMonths = 3 * 30 * 24 * 60 * 60 * 1000;
       var expDate = new Date();
       expDate.setTime (expDate.getTime() + ThreeMonths);
```

```
        SetCookie("UserName", document.MyForm.Name.value, expDate,"/",
"isrc.sandia.gov", null)
        SetCookie("UserEmail", document.MyForm.EMail.value, expDate,"/", "isrc.san-
dia.gov", null)
}
```

Setting document.cookie adds the data to the browser software's cookie database. It does not, as the syntax implies, overwrite the value of a variable named document.cookie. If the same variable is stored from two different contexts (web pages), then two copies of the variable/value page will be created.

Once stored, the browser exposes the variable/value pairs or cookie values as a single string containing all the variable/value pairs that are available within the context of the current page. If printed, the string might, for example, look like "UserName=Bob;UserEmail=bobbie;" However, because the escape function is generally used to store data, these values might not be in a readable form.

Retrieving cookies requires inspecting (parsing) the document.cookie string, looking for the first or all occurrences of the variable name string, and extracting the variable values. The following code can be used to extract the first occurrence of a named cookie string. It must be noted, however, that additional occurrences, if they exist, are ignored.

```
function GetCookie (name) {
  var result = null;
  var myCookie = " " + document.cookie + ";";
  var searchName = " " + name + "=";
  var startOfCookie = myCookie.indexOf(searchName)
  var endOfCookie;
  if (startOfCookie != -1) {
    startOfCookie += searchName.length; // skip past cookie name
    endOfCookie = myCookie.indexOf(";", startOfCookie);
    result = unescape(myCookie.substring(startOfCookie, endOfCookie));
  }
  return result;
}
```

As a close inspection of the code above reveals, two cookies are used in the comment book. One stores the reviewer's name and the other, their email address. These cookies get stored when the user clicks the submit button. This later automation is achieved by attaching the appropriate JavaScript function to the onSubmit keyword in the "FORM" tag.

```
<FORM NAME="MyForm" ACTION="AddComment.idc"
        onSubmit="saveUserID()" METHOD=POST>
```

Cookie values are retrieved while the comment page is being loaded by the browser. Just as the loader reaches the text input box for either the name or email, a line of JavaScript is executed to get the appropriate cookie and fill the blank in the form.

```
<INPUT TYPE="text" NAME="Name" VALUE="Anonymous" >
<SCRIPT language ="JavaScript"> <!-
document.MyForm.Name.value = GetCookie("UserName")
//-></SCRIPT>>
```

In a similar way to the reviewer name and email address values, a date value gets filled with the value formatNow() (which means build a date that represents now). The date function is obvious.

```
document.MyForm.Reference.value = WhereWasI()
document.MyForm.Date.value = formatNow()
```

3.3 Storing and Retrieving Comments.

Database interactions rely on standard HTML form SUBMIT (POST and GET) features, an SQL database system, and special automation functions from Microsoft's Peer Web Server software including ODBC, SQL, and dynamic HTML (HTX files).

The SUBMIT features in HTML (via POST or GET) provide two nearly transparent ways to send data to the web server. Figure 3 shows and numbers the sequence of events.

- (1) By clicking the SUBMIT link, a web request is made to the IDC file with data attached via the SUBMIT mechanism.
- (2 & 3) The Microsoft's Peer Web Server software recognizes these IDC files, launches a program module called httpodbc.dll through the Internet Server's API, and passes it to the IDC file handle and SUBMIT data.
- (4 & 5)Httpdodbc reads this data and forms an SQL request through the ODBC driver to the appropriate data source.
- (6 - 9) The results of the SQL query (6) are returned through the ODBC driver (7) and then formatted by httpodbc according to a custom HTX (8) file to produce a "regular" HTML file (9) that contains the formatted results of the web request.
- (10) Finally, the Peer Web Server returns this HTML file to the requesting browser (10).

The peer web server does not differentiate between data submitted by either the POST or GET methods. Thus, for the user, the major difference between POST and GET is whether the data is hidden or exposed to the user. In addition, because the "GET" method can only transfer 256 characters, "POST" is required for large data transfers. Names of data values are defined within the HTML code.

The IDC file, which Httpodbc.dll reads, contains names of an html template file (with an extension .htx) and an ODBC-compliant SQL database source, as well as an SQL query that is formatted in accordance with the

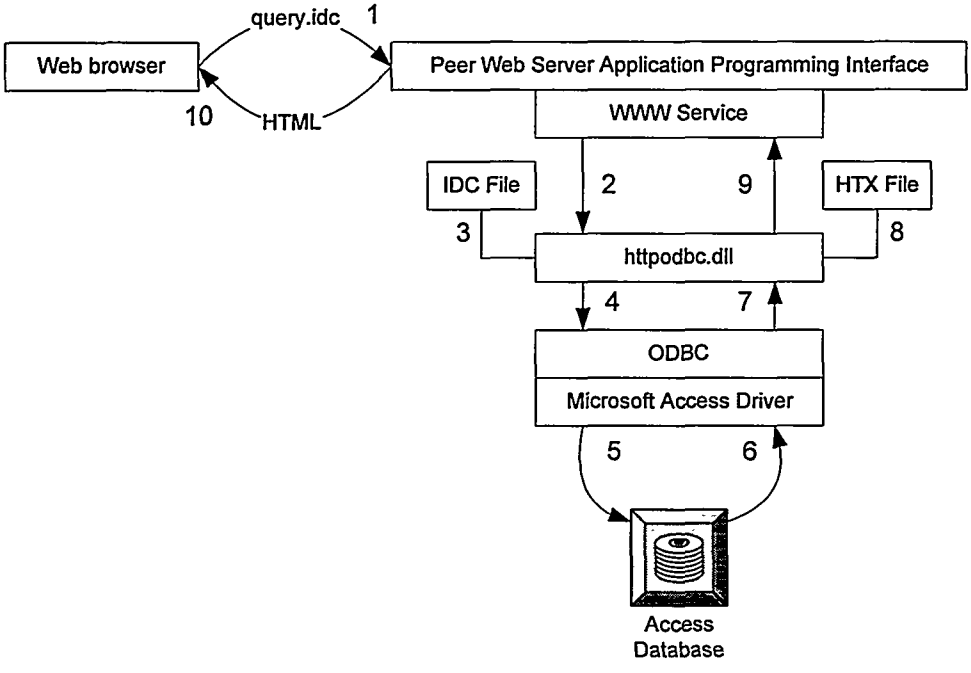

Figure 3: Web to Database Architecture

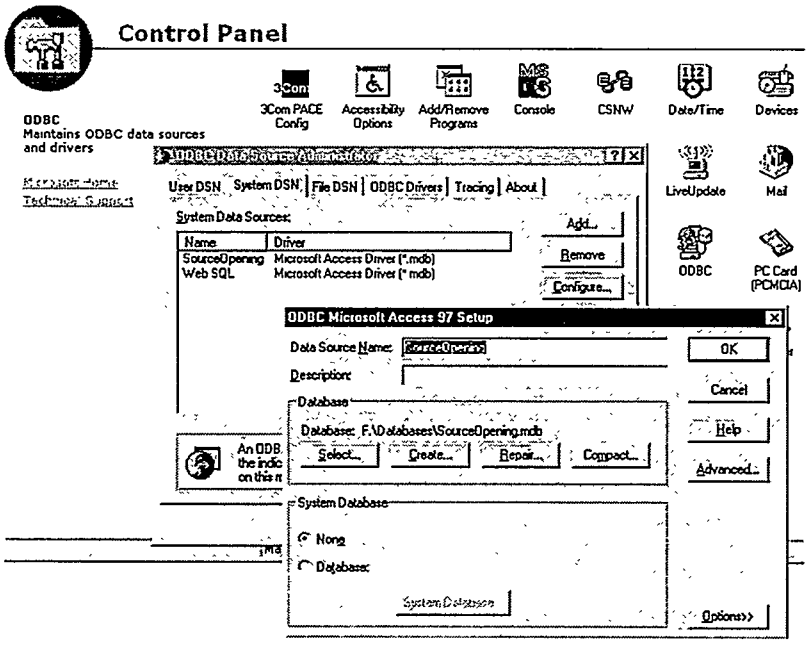

Figure 4: Connecting a Microsoft Access database to an ODBC data source.

database system (here, Access). Data transferred from the SUBMIT can be used as part of this query. The Httpdodbc.dll program reads the file, performs the SQL queries, and returns the results according to the htx file format.

For example, the following IDC file, named add_comment.idc, is used to store comment data from a SourceOpening datasource.

```
Datasource: SourceOpening
Username: sa
Template: AddComment.htx
RequiredParameters: Name
SQLStatement:
+   INSERT INTO Review
+   (Name, email, DateOfComment, Category, Reference, Comment)
+   VALUES('%Name%', '%email%', '%Date%', '%Category%',
+    '%Reference%', '%Comment%');
```

This file, while small, contains all the essential elements to translate the data in the web SUBMIT action into a database query, send the query to the right database, and return the results to a formatter file. The database query is formed as an SQL statement (last item), the database is referenced as the Datasource (first item), and the formatter file is referenced as the Template. (A username field is provided for high-end datasources.)

ODBC-compliant SQL data sources can be built using Microsoft Access. Alternatively, higher-end database tools like Microsoft SQL Server can be used for high-volume uses. The Access database files (.mdb) must be stored on the server and the ODBC control panel must be used to create a System data source and relate it to the database. (This is done by clicking the System DSN page, "adding" a service, and "configuring" it by selecting the appropriate database. The appropriate actions can be seen in Figure 4.)

While SQL statements are similar between databases, some differences exist. Thus, the SQL query must conform to the particular database (in this case Microsoft Access) being used. The most difficult issue of using

Microsoft Access as an ODBC data source is that the SQL interface is not widely documented. The book **Access 97 Developer's Handbook**, SYBEX press, is a particularly useful source of the needed information.

The datasource, SourceOpening, was built by creating a database file with Microsoft Access. The database has a table named Review with fields named Name, email, DateOfComment, Category, Reference, and Comment. (Note, SQL has a bug relating to using the word "date" as a field names.) This datasource was then published or linked on the database server by using the ODBC control panel and named SourceOpening. (See Figure 4.)

INSERT statements, as used in the add_comment.idc file, are fairly straightforward. The statement references the Review table and its fields and, in addition, references, through substitution, the values of data defined in the Make Comment Page's FORM INPUT fields (named Name, email, Date, Category, Reference, and Comment). The ODBC agent software (httpodbc.dll) substitutes the values of the data into the SQL call and then makes the call to the datasource.

In the Make Comment Page's case, the SQL statement performs an INSERT. That is, it creates a new record and stores the data supplied by the Make Comment Page. Because the INSERT command does not return data, the related template file, AddComment.htx, could be any html file. In this case, the file contains a simple JavaScript function that shows the user that the data was added and automatically backs up to the page on which the comment originated.

A more complex interaction can be found in the Report Query web pages/idl files. Here, search values are sent from the web page to form the search. The SQL statement (below) then extracts lists of values for Name, email, DateOfComment, Category, Reference and Comment.

```
+SELECT Name, email, DateOfComment, Category, Reference, Comment
+FROM Review
+WHERE Name LIKE '%Name%' and (Category = '%AuthEquipment%' or Category =
'%AuthProcess%' or Category = '%EquipmentCost%' or Category = '%AuthCostAnalysis%'
or Category = '%AuthGeneral%' or Category = '%AuthWeb%')
+ORDER BY DateOfComment DESC
```

The SELECT statement returns a series of data records from the database. These records are then formatted in accordance with a more complex htx file.

For example, the htx file used with the Report Query to build the Report Page looks like an html file with additional lines of code shown below. Here, the special tags <%begindetail%> and <%enddetail%> brackets the section of code that generates html for each record. Values for Name, email, DateOfComment, Category, Reference and Comment are inserted by using tags like <%Name%>. Additional HTML tags are used for all formatting. A conditional, <%if CurrentRecord EQ 0 %>, is used to break the loop. (Additional documentation on htx formatting can be found in chapter 8 of the peer web services documentation file /iisadmin/htmldocs/08_iis.htm.)

```
<%begindetail%>
<%if CurrentRecord EQ 0 %>
<%endif%>
<P>
<B> Name: </B> <%Name%> <A HREF=mailto:<%email%>> <%email%> </A>
<BR>
<B>Date </B> <%DateOfComment%>
<B>       Category:</B>  <%Category%>
<BR><B>Reference:</B> <A HREF=<%Reference%>> <%Reference%></A>
<BR><B>Comment: </B> <%Comment%>
<br>
<CENTER><A HREF = "/CommentBook/html/add_comment.htm?came from=<%Reference%>&">
Make Follow-Up Comment</A> </CENTER>
<hr>
<%enddetail%>
<%if CurrentRecord EQ 0 %>
```

```
<h2>Sorry, no entries match those criteria.</h2>
<hr>
<%endif%> 
```

Once formatted, the peer web server returns a regular-looking html file (the Report Page) to the user's browser. For example, the htx code fragment above generated the html code below for one particular data query.

```
<P>
<B> Name: </B> Michael J. McDonald <A HREF=mailto:mack@sandia.gov> mack@sandia.gov
</A>
<BR>
<B>Date </B> 1998-08-18 08:20:00
<B>       Category:</B>  General Comment
<BR><B>Reference:</B> <A
HREF=https://gisc.isrc.sandia.gov/~mack/ISRC_ONLY/Multimedia/html/deneb_to_movie.html
>
https://gisc.isrc.sandia.gov/~mack/ISRC_ONLY/Multimedia/html/deneb_to_movie.html</A>
<BR><B>Comment: </B> This page should say what the optimal size should be (either
490 or 480 pixels wide by about 370 high). Verify that this is a good size for scott
gladwell.
<br>
<CENTER><A HREF = "/GDPCommentBook/html/add_comment.htm?came
from=https://gisc.isrc.sandia.gov/~mack/ISRC_ONLY/Multimedia/html/deneb_to_movie.html
&">
Make Follow-Up Comment</A> </CENTER>
<hr>
```

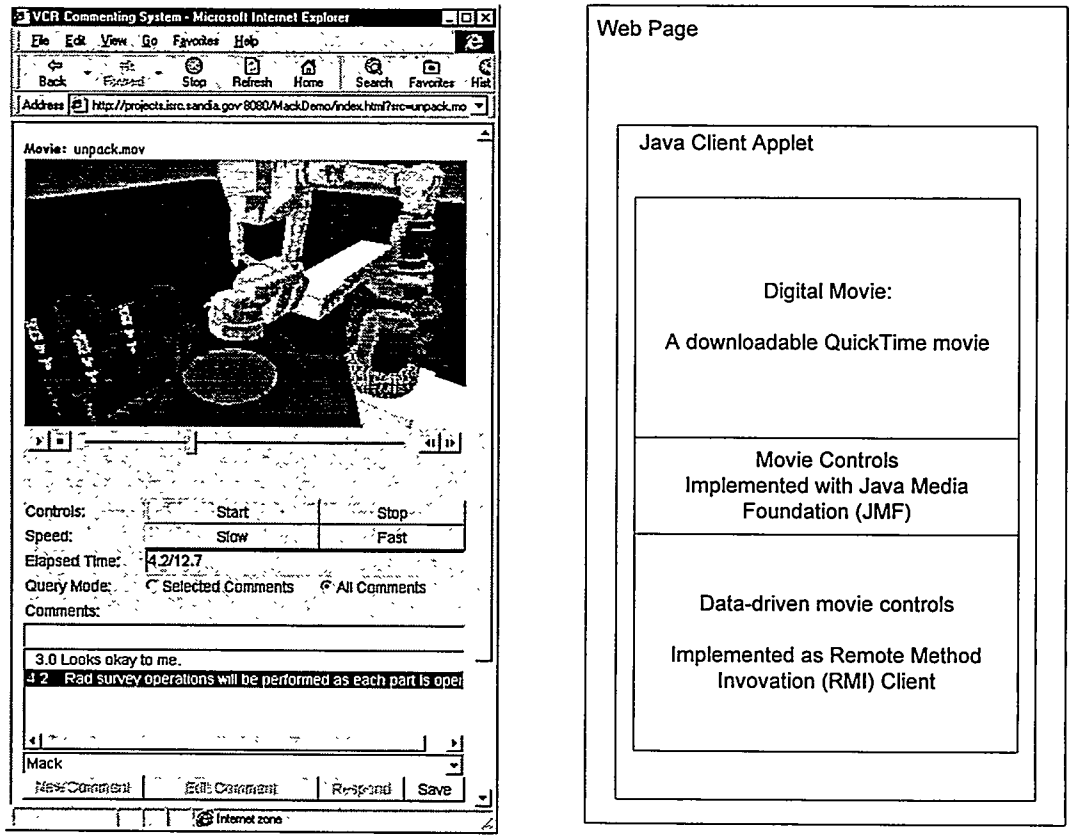

Figure 5: Sandia's Digital Movie Commenting System

3 Digital Movie Commenting

Sandia's digital movie commenting system appears as a movie player program with a few added controls, a text input area, and a scrolling text display area (see Figure 5). Clicking on the "regular" control buttons causes the movie to play, stop, or index as would be expected from any movie player program. Unlike a movie, clicking on text (usually in the form of comments) causes the movie to advance to a special place within the movie and adding text while the movie is paused causes the text to be added (along with identifying information) to the list of other comments. Finally, clicking a "Save" button saves the comments for others to review.

Like the page-by-page commenting system, the people who generally use this tool do so to provide or review detailed information about individual movies. Because the commenting system uses visual real estate, the tool is frequently used in a "stand-alone" manner, apart from the content pages. That is, the user clicks on a provide/review comments about movie link and is then brought to the web commenting page.

Within the commenting system application, the commenting area is active whenever the digital movie has stopped. A name text box allows users to use simple monikers to identify themselves and a commenting text box allows users to enter comments. When the user presses a return from within the comment box, the name and comment text "move" into a scrollable list of comments.

Within the Java code, the movie time is stored with each comment. This data is then reused, when the comment is "selected" by the user, to adjust the movie time to correspond to the stored values.

When the applet is loaded, an RMI-based interface is queried to populate the scrollable comment box with previously saved comments. Later, reviewers (users) are allowed to upload added comments to the remote database. This uploading function is provided through the same RMI interface.

The digital movie commenting system is built on several technologies (see Figure 5). Movies are created in any of several formats (Sandia generally uses QuickTime Cinepac format) and stored on a web server. A single Java applet is used to display the movie, provide controls, and interact with the remote database. The Java applet uses the Java Media Foundation (JMF) class to implement movie controls, a Remote Method Invocation[3] (RMI) interface to a remote database to interact with the database, and standard awt Java widgets package for other controls. An RMI-based server is used to communicate with the movie commenting system client software and the Java ODBC package is used to let the server interact with an SQL-based database (developed, for example, in Microsoft Access). A web page (HTML) is defined to define the page's layout as well as pointers to the movie source and java class files.

3.1 Connecting the Applet to a Database through the Network

Two different networking protocols are used to support the movie commenting system (see Figure 6). The first is standard Hypertext Transfer Protocol (HTTP) which is used to serve the web page, the movie file, and the Java client applet. The second protocol, RMI, is used to send and retrieve individual comments between the Java client applet and a Java server applet. Within the server computer, the server program communicates with a database via SQL calls made through an ODBC service.

For Java-related security reasons, RMI client applets are typically loaded from the same host computer that the server applet is run from (this allows users to prevent, for example, remotely served Java applets from communicating with corporate Intranet data sources). Thus, the RMI server typically runs alongside a web server. Our implementation uses windows to allow the RMI server program to communicate with an SQL database through an ODBC service. Conversely, for other security and efficiency reasons (i.e., because NT is a limited, less reliable OS), it is often beneficial to load the remaining web content from a second web server. This is particularly true when the movie content is part of a larger web site or is being served in a secure mode.

The RMI has a unique way of working across firewalls. First it attempts to communicate via a normal socket (over the REGISTRY_PORT which has a default value of 1099). Next, it tries to use the user's HTTP proxy

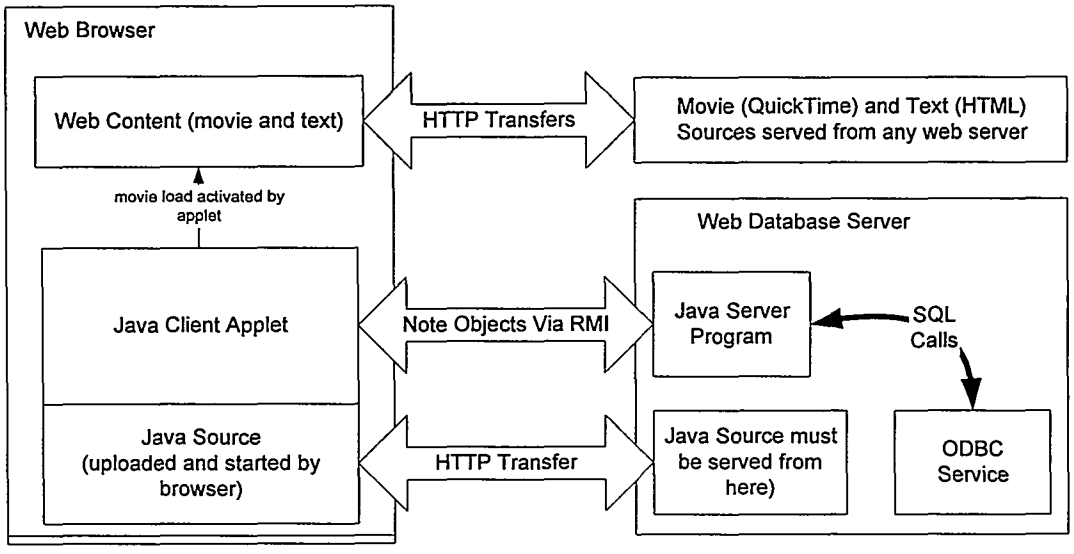

Figure 6: Network Diagram of Sandia's Digital Movie Commenting System

(on port 80) to forward packets to the registry port. Finally, it attempts to execute an HTTP post method (over port 80). The server computer must have its registry port and should have port 80 directly accessible to customers. Router-based firewalls (especially Sandia's ISRC EON network) are often configured to pass port 1099 communications by default but block port 80. Special arrangements may need to be made to provide the widest-possible access to the RMI-based application. In addition, it should be remembered that some firewalls (especially Sandia's IRN firewall) do not allow this, or any Java-based application to be downloaded from Internet or Sandia-EON site. As a result, this technology cannot be made accessible to these users.

The networking portion of Sandia's commenting system's use of the RMI closely follows Sun's "getting started" documentation available at http://java.sun.com/products/jdk/rmi/index.html. A server interface is defined as an extension to remote. This server interface is implemented in a second class that extends the java RMI UnicastRemoteObject. This server implementation is run on a web server as a java application (i.e., run within java, and not as an application). Finally, the movie player program (which runs in the user's browser) communicates with the server applet over the RMI interface.

Unlike Sun's demonstration code, our server implementation includes java SQL functions to query and store data in an SQL database that is accessed through the ODBC interface. In addition, a serializable "Note" class has been defined as a vehicle for storing data and transfer it between the client to server.

3.2 Applet Support for Movie Playback

Movie playing functions are supported through the emerging Java Media Framework API (JMF).[4] JMF specifies a simple, unified architecture, messaging protocol, and programming interface for media players, media capture, and conferencing. JMF 1.0 supports the synchronization, control, processing, and presentation of compressed streaming and stored time-based media, including video and audio.

The commenting system relies on JMF's ability to play digital movies at the same efficiency as native viewers (i.e., the QuickTime plug-in) while providing hooks that allow the application to seek to a new position in the video file and initiate playback.

Each JMF Player has a Time Base that defines the flow of time for that Player. Within the commenting applet,

Player's media time (which represents the current position in the media stream) is stored with each comment. When a Player is started, its media time is mapped to its time-base time. When a specific comment is selected, the media time is advanced to the stored value.

According to the JMF documentation,[4] "Java Media Player applets can be in one of six states. A Clock interface defines the two primary states: Stopped and Started. To facilitate resource management, the Controller breaks the Stopped state down into five standby states: Unrealized, Realizing , Realized, Prefetching, and Prefetched."

- In normal operation, a Player steps through each state until it reaches the Started state:
- A Player in the Unrealized state has been instantiated, but does not yet know anything about its media. When a media Player is first created, it is Unrealized.
- When Realize is called, a Player moves from the Unrealized state into the Realizing state. A Realizing Player is in the process of determining its resource requirements. During realization, a Player acquires the resources that it only needs to acquire once. These might include rendering resources other than exclusive-use resources. (Exclusive-use resources are limited resources such as particular hardware devices that can only be used by one Player at a time; such resources are acquired during Prefetching.) A Realizing Player often downloads assets over the net.
- When a Player finishes Realizing, it moves into the Realized state. A Realized Player knows what resources it needs and information about the type of media it is to present. Because a Realized Player knows how to render its data, it can provide visual components and controls. Its connections to other objects in the system are in place, but it does not own any resources that would prevent another Player from starting.
- When prefetch is called, a Player moves from the Realized state into the Prefetching state. A Prefetching Player is preparing to present its media. During this phase, the Player preloads its media data, obtains exclusive-use resources, and anything else it needs to do to prepare itself to play. Prefetching might have to recur if a Player's media presentation is repositioned, or if a change in the Player's rate requires that additional buffers be acquired or alternate processing take place.
- When a Player finishes Prefetching, it moves into the Prefetched state. A Prefetched Player is ready to be started; it is as ready to play as it can be without actually being Started.
- Calling start puts a Player into the Started state. A Started Player's time-base time and media time are mapped and its clock is running, though the Player might be waiting for a particular time to begin presenting its media data.

The Player posts TransitionEvents as it moves from one state to another. The ControllerListener interface provides a way for the applet to determine what state a Player is in and to respond appropriately. Using this event reporting mechanism, lets the applet to ensure that the Player is in an appropriate state before calling methods on the Player.

4 3D Animation Commenting

Sandia's 3D animation commenting system appears and works much like the digital movie commenting system. Here, however, the graphical animation is played in a Virtual Reality Modeling Language (VRML) browser while the controls and comment editing areas appear in a separate Java applet window.

With Sandia's commenting system, the VRML animation is controlled much like any digital movie. In addition, the VRML scene can be navigated (i.e., the viewpoint can be changed) as with any VRML animation. Like the digital movie commenting system, users can stop the animation at any point in time, add comments, and have the comments stored in a central repository. Unlike with digital movies, clicking on the comments advances the viewpoint along with the time point in the animation.

The 3D animation commenting system is built on several technologies. Some of these technologies are only supported through proprietary software environments, others are more general. In all cases, the general, extensible principals were used to allow adopting emerging VRML browsers and programming features. These technologies include

1. VRML and browser-embedded VRML browsers.
2. A Sandia-developed way to use of VRML's key frame animation features.
3. Cosmo 3D's EIA interface to VRML to allow setting various parameters in the VRML environment from a Java applet.
4. The RMI-ODBC-SQL system used in the digital movie commenting system.

VRML (often pronounced 'vermal') is the file format standard for 3D multimedia and shared virtual worlds on the Internet. VRML provides, structured graphics, and extra dimensions (depth and time) to the online experience. VRML applications range from business graphics to entertaining web page graphics, to manufacturing, scientific, entertainment and educational applications, to shared virtual worlds and communities. Since its inception in 1994, VRML has been an open standard. An ISO Standard VRML97 has been approved. The next generation of VRML is under development.

VRML files contain geometric, animation, and program features. For example, a VRML file might define how geometric models should move in response to user interactions. These VRML files are played in VRML browsers that are typically run within web browsers. In particular, the Cosmo 3D player,[6] required for Sandia's commenting system, runs from within Netscape and Explorer 4.0 browsers.

Sandia's commenting system uses VRML files that are mainly built on general modeling and key-frame animation approaches. Models are constructed with kinematic structures that correspond to the simulated environment. As the simulation progresses, device and joint position and orientation changes are stored as key frames. These files are described in Section 4.1.

Sandia's comment recording application is written as a Java applet that communicates with a VRML browser and a Java RMI-based database (like that described in Section 3). Both the applet and VRML browser are, in turn, run inside a web browser. The Java to VRML communications are described in Section 4.2 and the integrated system is described in Section 4.3.

4.1 VRML Model Construction

Sandia's VRML models are typically automatically produced from simulation or other CAE software tools. As of this writing, Sandia has developed complete translators for Deneb Inc.'s Envision (a robot simulation and Virtual Reality application development package) and Sandia's Umbra (a simulator developed to support telerobotics, vehicles, small smart machine systems and machine simulator development) codes. In addition, Sandia has developed a partial translator for its Archimedes assembly analysis code.

Typically, these codes use 3D models for both simulation and visualization. The simulators compute the positions and possibly shapes of various geometric elements on a continually increasing time base. While the simulations only compute position of elements at specific time steps, the underlying assumption of the simulations is that the real devices operate on a continuous time base. Due to the geometric nature of the environments being simulated, a significant portion of the simulation results can be conveyed as 3D graphical animations.

To enable the commenting system, Sandia's VRML files are constructed with a few specific features. Specific objects or nodes are defined and these nodes must be connected or routed according to standard VRML practices. Three specific nodes, a Time Sensor, a Scalar Interpolator, and a Proximity Sensor are used as control nodes for animation control.

Like all animations, a Time Sensor is used to provide a time base and duration for the animation. For synchronization, a single time base is used. Next, a Scalar Interpolator with a 1:1 interpolation is used to provide a hook for driving the simulation from the Java applet in VCR-like fashion and advancing the simulation to specified time points. A Proximity Sensor with an attached Viewpoint is used to provide a hook for recording and re-using viewpoints through the Java applet. Motions are defined as key frame animations in terms of Position and Orientation Interpolators while color and transparency changes are defined with Scalar Interpolators. A Touch Sensor or equivalent node is used to allow the user to initiate animation by clicking on any graphic element.

The elements are routed or connected as shown in Figure 7. The Touch Sensor is connected to the Time Sensor to provide a mechanism for starting the animation clock. The Time Sensor's time base (fraction changed) is routed to the 1:1 Scalar Interpolator and, in turn, its output (value changed) is routed to each of the Position, Orientation, and Scalar Interpolators. (Each Interpolator has a key frame/value pair corresponding to each time the corresponding geometric property changed.) Finally, the outputs (value changed) of these interpolators are connected to the various geometric properties (i.e., set translation, rotation, transparency, and color.) The code for the routing operations appears as follows.

```
DEF CLOCK_TIME TimeSensor {
  cycleInterval 32.9
}

DEF MOVIE_TIME
ScalarInterpolator {
  key [0.0 1.0]
  keyValue [0.0 1.0]
}

ROUTE
CLOCK_TIME.fraction_changed
TO MOVIE_TIME.set_fraction
```

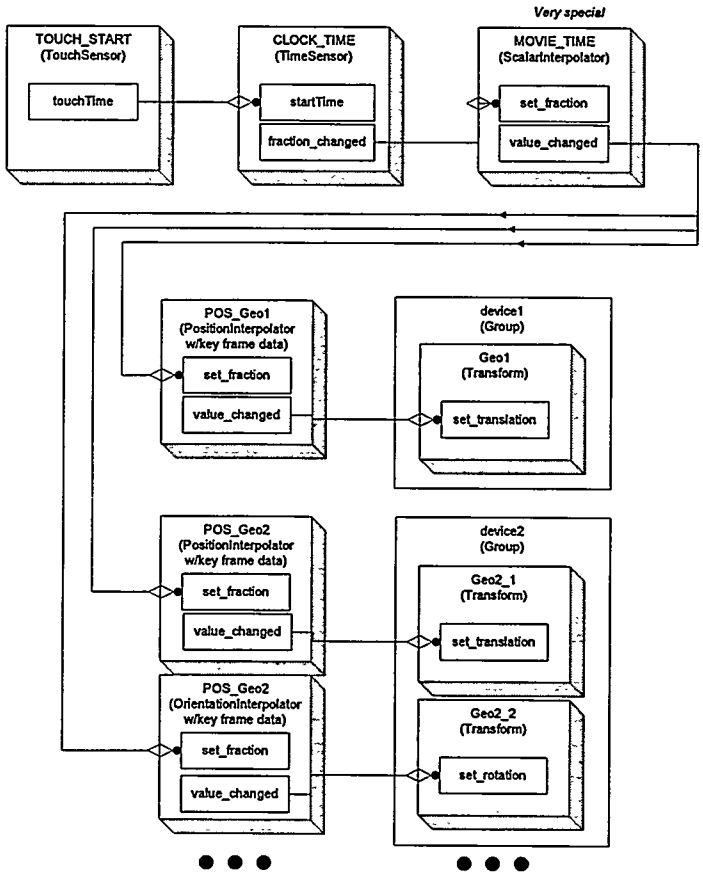

Figure 7: VRML routing required for animation control.

When the animation is loaded into a standalone browser, activating the touch sensor

causes the animation to play. That is, the animation starts when the user "clicks" on any geometric element in the scene. The activation causes the touch sensor to look up the current clock time and pass that value into the Time Sensor's Start Time. The Time Sensor then begins to compute its fraction changed value according to the formula: f = fractionalPart((*time* - startTime) / cycleInterval) until it reaches 1 (where it stays).

This fraction changed is passed to the first Scalar Interpolator, which passes its input value, now unchanged onto the various (i.e., scalar, rotation, or visual) interpolators. These Interpolators then use their key frame data to compute geometric properties (position, orientation, and visual) for the various geometric elements. The elements receive this data and move or change their displays in the world model.

It is worth noting, at this point, that due to the interpolation scheme used here, geometric motion does not identically replicate simulated motion. The simulator computes geometric positions at times T1, T2..., Tn and generates key frame data for these points in time. The VRML clock computes the time at slightly different intervals and computes interpolated positions for the geometric elements. (The VRML browser sets these intervals to make the total simulation take the same amount of time to play back regardless of the speed of the playing computer.) The result is that the motion plays more smoothly but less accurately. In Sandia's application, the tradeoff is desirable, as the visual quality is very important. In other applications, additional key frame data can be added to keep the motion jerky but accurate. For example, the Position Interpolators could be made to produce stair-stepped transfer functions by repeating position at the beginning and end of each time step.

Besides animation, a small set of code is used to define VRML view points and sets up the VRML model for viewpoint tracking. Like the first scalar interpolator, the features for view point tracking have little, or no functionality in the stand-alone VRML animation. Conversely, they have everything to do with the commenting system described in the next section. The code is shown below. Its use is described in section 4.2.

```
Group {
  children [
    DEF PS ProximitySensor {
       center 0 0 0
       size 10000 10000 10000
    }
    DEF DENEB Viewpoint {
      position 0 0.924858 2.79959
      orientation 0 0 0 4.21468e-08
      description "last view "
    }
  ]
}
```

4.2 Controlling VRML Models though the EIA

Beyond their use as stand-alone animations, these VRML animations can be played under the control of other programs. The Cosmo Player VRML 2.0 browser, used in the commenting application, includes an External Authoring Interface (EAI) that allows developers to easily extend the functionality of and thereby build commercial and custom applications incorporating 3D, dynamic content. In essence, the EAI provides a method for developing custom applications that interact with, and dynamically update a 3D scene. These outside applications "talk" to the VRML scene.

In the Java EIA interface, browser data is accessed through the vrml.external package. In building EIA applications, the browser data structure is accessed and used to retrieve various VRML nodes. From this, the nodes are accessed to retrieve various event interfaces within the nodes. These events are then exercised to store or extract data within the VRML scene. Figure 8 shows an example of this duality between VRML and Java developed for this application.

Communicating with a particular node is done by first getting a reference to the node, using the getNode( ) method. This method is passed a string corresponding to the DEF name of a node in the root VRML scene. Only named nodes in the root world (the part of the world loaded from the HTML page) can be accessed from the EAI. This can be done, for example, with the instruction:

```
root = browser.getNode("ROOT");
```

The root variable now contains a reference to the node named ROOT in the scene. With this reference to a node, Java methods can send events to its eventIns, get the current value of its eventOuts, or register methods to be called when the eventOuts send events. To send an event, methods first get a reference to an eventIn of the node using the getEventIn( ) method, then send events using the setValue( ) method. To receive events (query the system for values), methods get a reference to the eventOut of the node using the getEventOut() method, then query for events by using the getValue() method. For example:

```
class ScalarInterpolator
  {
    Node _node;
    EventOutMFFloat _keyValue_changed;
    EventInSFFloat _set_fraction;
       …
       // b is the browser (from a Browser.getBrowser(this) )
       // label is the name of the scalar interpolator (i.e., MOVIE_TIME)
    ScalarInterpolator(Browser b, String label)
    {
      _node = _browser.getNode(label);
      _keyValue_changed = (EventOutMFFloat) _node.getEventOut("keyValue_changed");
      _value_changed = (EventOutSFFloat) _node.getEventOut("value_changed");
…
    }
    // This method lets you set the movie fraction while the clock is stopped
    void setFraction(float v) { _set_fraction.setValue(v); }
    // the set and get key and KeyValue methods let you invert the movie
    float[] getKeyValue() { return _keyValue_changed.getValue(); }
…
```

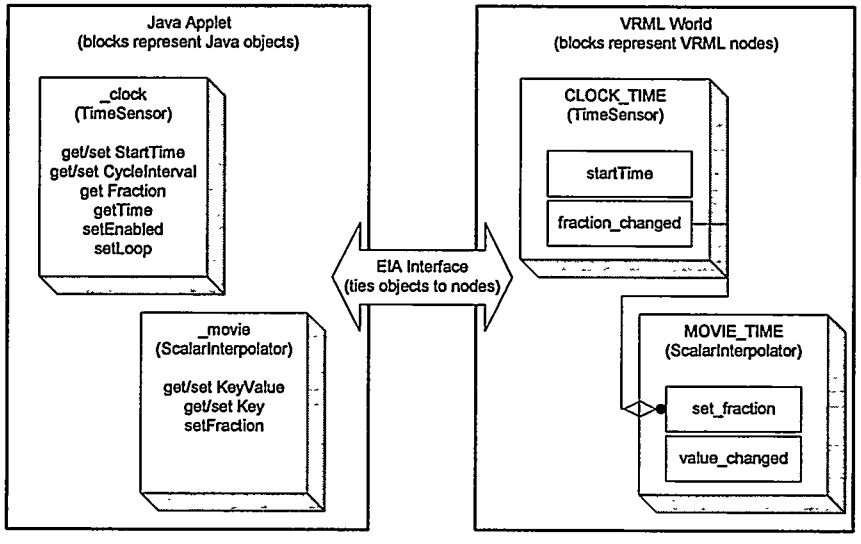

Figure 8: VRML to Java Mapping via EIA Interface

Animation timing or playback can be controlled through the control nodes described in section 4.1. To provide data hiding, Java classes are defined to correspond to the VRML Time Sensor, Scalar Interpolator, and Proximity Sensor. Instances of these classes (a clock, a movie time, and a view) are created to correspond to each of the control nodes.

The Time Sensor contains methods to get and set the simulation clock's Start Time and Cycle Interval as well as methods to get the Current Time and the Fraction of the movie that has played, and methods to set the clock's Enabled and Loop properties. The animation is started and stopped by adjusting the clock's Start Time. (The animation starts when the Start Time is greater than the current Time.) The animation playback speed is adjusted through the Cycle Interval. The Fraction is used to determine whether the movie has finished playing and the Time is used to compute an appropriate value for starting and stopping the animation. Setting Enabled to false stops the movie. The Loop value can be used to cause the animation to automatically loop (rather than play until finished) and is also used as part of a trick for querying the VRML world for the current time.

The Scalar Interpolator contains methods to get and set the Key and Key Value pairs as well as methods to set the Fraction value. Key Values are stored as a vector (1 dimensional array). Reversing the order of the Key Values causes the movie to play backwards. That is, it causes the Scalar Interpolator to output a scaled time value which is 1 – the clock's time. This time scaling must accompany a changing of the fraction that the clock is providing. For example, if 25% of the movie has been played forward, then 75% remains to be played backwards.

In addition to reversing the movie, the Scalar Interpolator provides a mechanism for stopping or stepping the animation at a given point in time. Here, the clock is stopped and the Scalar Interpolator's Fraction is set directly from the animation. Animations can be slowly stepped by adjusting this fraction value. This ability to directly set the fraction is not, however, used as a substitute to using the Time Sensor, as the EIA cannot provide as smooth and efficient of updates as the VRML world can provide.

In VRML (as with most 3D animation systems), scenes are rendered with respect to the theoretical position of an eye placed in the world at a specific orientation. These are called viewpoints. In the commenting tool, A ViewPort and a View object are used to record viewpoints and later set the view position to earlier recorded viewpoints. A ViewPort java object is connected to the Proximity Sensor (here named PS) and a View object is connected to the viewpoint that is within the Proximity Sensor (here named DENEB). The ViewPort object provides (through the EIA) the ability to query the current viewpoint's position and orientation. Later, the View object is used to set the viewpoint to past saved viewing positions.

4.3 Adding and Reviewing Animation Comments

Sandia's VRML animation commenting system, shown in Figure 9, has a similar interface and is used for similar reasons as the digital movie commenting tool. Clicking on the "VCR-like" control buttons causes the movie to play, stop, or indexed as would be expected from any movie player program. In addition, clicking and dragging within the scene window (according to interface practices developed by SGI) allows the user to adjust the viewpoint to any position and orientation. The user can change this viewpoint at any time (while the animation is playing or stopped). Adding text while the animation is paused causes the system to record the text as a comment. Later, clicking on comment text causes the animation to advance the time and the viewpoint to adjust to be in the place from which the comment was added. Finally, clicking a "Save" button saves the comments (across the network) for others to review.

VCR-like recording functions are implemented with the Time Sensor and Scalar Interpolator objects described above.

- A "Play" button action is implemented by querying the Current Time and then setting the Time Sensor's Start Time to be greater than the current time (see stop button). If the animation has been

partly played, this start time is set so that it will restart the animation with any stored "Current Fraction" values (see Stop button). This button then turns into a "Stop" button.

- The "Stop" button is implemented by saving the Current Fraction in a local variable and setting the Start Time to zero.
- A slider widget is used to let the user step the movie slowly or drag the time to any point within the movie. This widget performs this action by stopping the animation (if running) and then adjusting the Scalar Interpolator's Current Fraction value.

The commenting area is active whenever the animation has stopped. A name text box allows users to use simple monikers to identify themselves and a commenting text box allows users to enter comments. When the user presses a return from within the comment box, the name and comment text "move" into a scrollable list of comments.

Within the Java code, animation time and viewpoint data are stored with each comment. This data is then reused, when the comment is "selected" by the user, to adjust the animation time and viewpoint to correspond to the stored values.

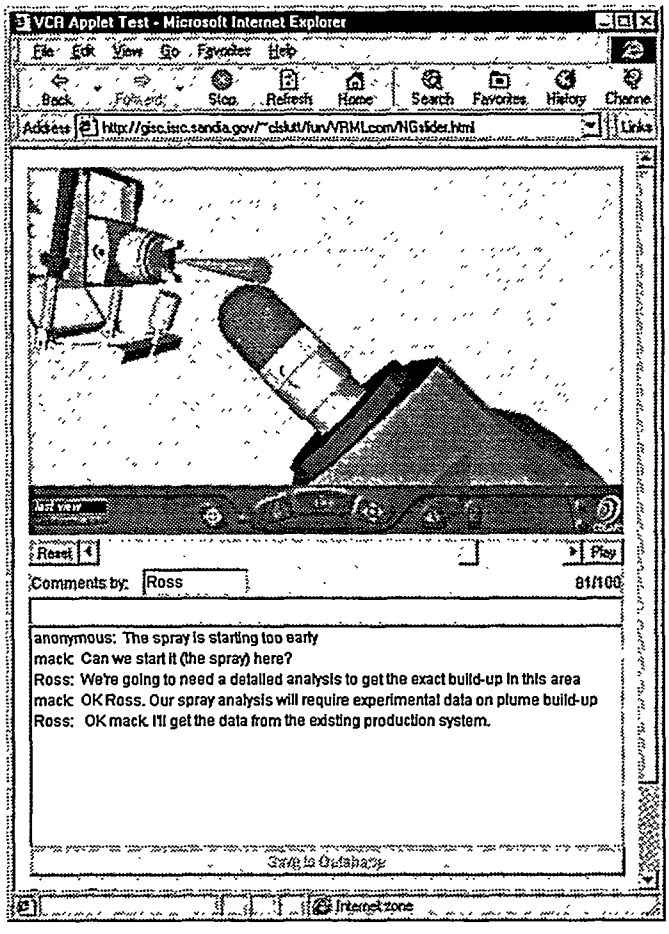

Figure 9: VRML Commenting System User Interface

When the applet is loaded, a database is queried via an RMI server, to populate the scrollable comment box with previously saved comments. Later, reviewers (users) are allowed to upload added comments to the remote database. This uploading function is provided through the same RMI interface. Code for this function is nearly identical with that described in Section 3. In our implementation, the same server code and database are used for both applications.

5 Testing

The technologies described here have both been tested and validated functionally and tested by expert user groups for appropriateness. In addition, they are beginning to be deployed in practical engineering processes.

The web page feedback system was first tested on a medium-sized engineering project that developed and analyzed two alternative robotic system designs for a uranium hexaflouride shipping container decontamination. Here, web pages were used to present information and review information was collected through the web feedback system (described in Section 2). In addition, VRML models were used to animate various solution concepts. In this deployment, engineers in the field provided review information that was then used by an on-site engineer to correct and update the published information. In addition, the need to have follow-up links

connected to review comments (rather than referencing web page) was discovered and implemented.

Additional testing of the revised web page feedback system was performed on a variety of projects. Currently unresolved issues include better management and easier separation of comments between projects that utilize the same database. Here, it has been found that it is beneficial to use one database to support several projects. Additional coding needs to be implemented to separate comments between projects.

Deployment of the movie and VRML animation commenting tools has been hampered by the fact that these two systems relied on new software tools that have not yet been widely deployed. (While these tools are freely available on the web, many potential users have not installed them on their computers.) Problems included

- Firewall blocking of Java: throughout the study time, software had been installed on Sandia's firewalls to block Sandians from accessing Java applets.
- Reliance on Internet Explorer 4.0: The Netscape products could not run our java applets. Sandia has scheduled its Internet Explorer 4.0 deployment for CY 99.
- Uncertainty of Cosmo VRML browser. In spring, 1998, SGI, who developed Cosmo, had formed a separate company, Cosmo Software, to develop and sell the Cosmo browser. By summer, 1998, SGI dissolved the company. In the fall of 1998, the Cosmo libraries were purchased by Platinum Technology, Inc., who announced plans to merge the Cosmo product with its existing VRML browser. These uncertainties have slowed the wide-spread deployment of the VRML browser and slowed the expected release of a stable browser for Macintosh platforms.

It is expected that usage will increase as these deployment issues are resolved. Thus, market maturity appears to be the main limiting factor.

6 Conclusions

In conclusion, engineering projects generate heterogeneous collections of data that must be reviewed by a diverse set of customers and peers. Due to the complexity of the information generated and shared, it is becoming necessary to electronically present this data, rather than rely on traditional (i.e., paper) forms. Sufficient electronic presentation technologies are only now becoming sufficiently advanced to support individual presentation needs. In addition, web-based technologies have potential to provide the glue to bind a large diversity of engineering information into unified presentations.

The problem of electronic presentation begs solving the problem of electronic review. To whit, data that is too complex to present in paper form is often too complex to review in paper form. This paper presented three prototype technologies for enabling the needed review. The first system ties comments to specific web pages, the second system ties the comments to specific frames of digital movies, and the third ties the comments to specific times and viewpoints within 3D animations. Each allows people to make context-sensitive comments about specific web content and electronically ties the comments back to the web content being referenced. All strongly rely on standard web software tools and development approaches.

7 End Notes

1 Computer Aided Design Report, Vol. 18, No. 4, April 1998,

2 With HTML, for example, content can be reused in several places in a report without requiring that it be duplicated within the original files. This can be significant, for example, if the same movie is used on two different "pages" of a report. In addition, HTML provides more natural mechanisms for separating content on a page-by-page basis. This can be significant, for example, if several movies are used within one report.

3 Java Remote Method Invocation Specification, Copyright 1997, Sun Microsystems (available at ftp://ftp.javasoft.com/docs/jdk1.2/rmi-spec-JDK1.2.pdf)

4 The JMF 1.0 API was developed by Sun Microsystems, Inc., Silicon Graphics Inc., and Intel Corporation. Sandia's commenting system uses Intel's version of the API.

5 VRML Consortium (Web 3D Consortium) sources. Available at http://www.vrml.org.

6 Cosmo 3D was developed by Silicon Graphics, was used to form the startup company Cosmo Software Inc., and, more recently, was purchased by Platinum Technology. The software can still be downloaded from . Version 2.1 was used for this development effort.

Internal Distribution Only:

1	MS9018	Central Technical Files, 8940-2
2	MS0899	Technical Library, 4916
1	MS0619	Review and Approval Desk, 15102 for DOE/OSTI
1	MS0161	Patent and Licensing Office, 11500
1	MS0188	C. E. Meyers, 4523
1	MS1002	P. J. Eicker, 9600
1	MS1004	R. W. Harrigan, 9623
1	MS1004	P. Bennett, 9623
1	MS1004	S. Gladwell,9623
1	MS1004	E. Gottlieb, 9623
1	MS1004	J. M. Griesmeyer, 9623
1	MS1004	M. J. McDonald, 9623
1	MS1004	F. Oppel
1	MS1004	C. Slutter, 9623
1	MS1004	ISRC Library
1	MS1008	J. Fahrenholtz, 9621

Second Printing, October 2000

Distribution:

1	MS9018	Central Technical Files, 8940-2
2	MS0899	Technical Library, 9616
1	MS0612	Review and Approval Desk, 9612 for DOE/OSTI
1	MS0161	Patent and Licensing Office, 11500
1	MS1004	M. J. McDonald, 15221
5	MS1004	ISRC Library