DISTRIBUTED DESIGN COLLABORATIONS EMBEDDED IN THE WEB

Brian F. Loss* , Richard F. Riesenfeld, Russell D. Fish
Department of Computer Science
University of Utah
Salt Lake City, UT 84112
bfl@eai.com rfr@cs.utah.edu fish@cs.utah.edu

Abstract

We describe two components of an effective distributed engineering design collaboration environment which we are developing. The system framework incorporates support for the whole process of manufactured product design from early concept through detailing, analysis, and manufacturing. An object-oriented design representation is provided by the Alpha_1 design and process planning research system. The system is augmented by new components, Model Views *and an* Alpha_1 Web Browser Plug-in, *which incorporate a full CAD capability into familiar WWW browser environments. Objects visible in Model Views can be hyper-linked in much the same way as HTML documents. In this way, we achieve full access to a 3D CAD system, including model creation and modification within web pages, in addition to the 2D view-only mode typical of present web page presentations.*

Introduction

An engineering design environment capable of representing and facilitating the whole design process and allowing designers to collaborate across large distances during the design would have a number of benefits. For example, experienced designers are a rare commodity. Designers can enjoy more convenient and more frequent access to design specialists if the two groups can collaborate without having to be physically together. Representing the whole process provides a record of the process, decisions, alternatives, etc. that led to the final design. The advantage of this information over a CAD model or series of blueprints is clear. This kind of integrated support for the entire design process is still lacking in state of the art CAD/CAM products [7,9].

Our first step towards achieving these goals is an effective distributed engineering design collaboration environment, the Design Web. The Design Web will incorporate support for the whole process of manufactured product design from early concept through detailing, analysis and manufacture. (Figure 1.) The 3D CAD system which provides an object-oriented design representation for the Design Web, is the Alpha_1 design and process planning research system developed at the University of Utah.

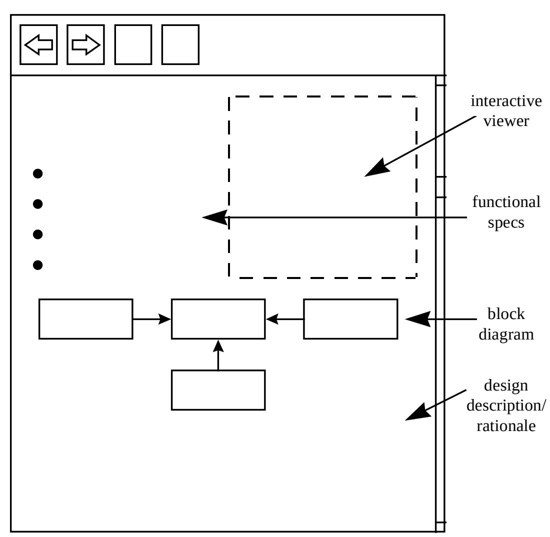

Figure 1: Schematic of a Design Web page

* Current Affiliation: Engineering Animation, Inc., Salt Lake City, Utah

The Design Web has an architecture that will support free transition between the view-only mode typical of present web page presentations and Alpha_1's model creation and modification capabilities, thereby incorporating full access to a 3D CAD system within a familiar WWW browser environment.

Our goal is to contribute to the understanding of the next generation of distributed collaborative CAD systems by reporting on two additions to the Alpha_1 system: *Model Views*, and *an Alpha_1 Web Browser Plug-in*. These additions allow designers access to 3D CAD model creation and modification capabilities from within web pages as the pages are dynamically authored during the distributed design process.

Background: Alpha_1

Alpha_1 is a parametric CAD system with integrated feature-based CAM support developed at the University of Utah [2]. Software support allows the user to automatically generate and simulate manufacturing code from process plan feature object models for computer numerically controlled machine tools. Alpha_1 incorporates a client-server architecture (Figure 2), based on processes connected by streams of relocatable persistent objects. This enables both a central representation of the design model for consistency, and distributed 3D display and modifications to be made to the model for high user interface performance.

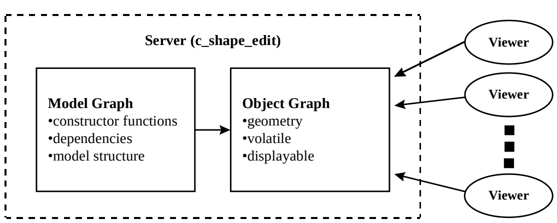

Figure 2: The Alpha_1 client-server architecture

The underlying representation for Alpha_1 models is a model object graph. It is a high-level representation, where models are represented as a dependent collection of manufacturing features, NURBS sculptured solids and underlying engineering and shape calculations, rather than just polygons. The model graph also maintains the dependency information necessary for Alpha_1 to implement parametric models. Actions such as drawing, picking, and NC code generation are supported via traversals of the model object graph. Properties inherent to the model (e.g., color and texture) are stored using the model attribute capabilities of Alpha_1 objects. Other model display properties that are specific to a window are stored with the window object itself, and are accessed via a table that associates the properties with the model objects visible in the view.

Additions to Alpha_1

Working toward an integrated engineering design environment for distributed collaboration, we have augmented Alpha_1 by two new components that incorporate a full CAD capability into a WWW browser environment. The additions are *Model Views* and an *Alpha_1 Web Browser Plug-in*.

A model view provides persistent memory for an active model viewing client window, saving state information relating to visible objects, view point and scale factor, and display properties in the same way as other Alpha_1 model objects. A hierarchy of linked model views represents structural relationships in the evolving product design and allows different members of a collaborative team can each simultaneously interact with different views of the model. The Alpha_1 Web Browser Plug-in component is based on the Netscape™ plug-in architecture, which allows Alpha_1 client programs that utilize the Tcl/Tk user interface toolkit to be embedded in HTML documents. The design process representation is composed of structured descriptive text and equations, together with embedded figures of any type supported by the Browser. These extensions to Alpha_1 are discussed below in more detail.

1. Model Views

Existing 3D CAD display clients, including those of Alpha_l, are quite capable of displaying a model of the product being designed. However, it is not sufficient to simply embed such a viewer, along with the associated Alpha_1 model object graph, within a Design Web document. A model embedded in the interactive distributed engineering environment envisioned in the Design Web will usually have some additional restrictions placed on it. For example, to protect proprietary information the designer may want to show only certain parts of the model to certain participants in the collaboration. Or, in a different case, the document may contain information explaining a portion of the model, and the picture should be from a specific viewpoint as a result. Perhaps the description refers to the highlighted portion of the model. Thus, embedded models in a Design Web document require additional information that is not specific to the model itself. Display properties such as the current viewpoint, what parts of the model are selected, highlighted, or visible, and so on apply only to the model when viewed in this specific figure within a page of a Design Web narrative. One method a designer could employ to capture this information is to run a CAD viewer, set up the viewpoint and display qualities as desired, and capture a screen dump for use in the Design Web document. Although feasible, this approach does not yield a "live" model embedded in the document.

The ability to interact with models from within a document is an important aspect of what we want to achieve in the Design Web. Many documents will contain information, such as descriptions of certain portions or qualities of a design, that dictate a particular presentation of the model. However, such a presentation is only a point of reference to provide a context for the non-geometric information in the document. The user can also benefit from being able to interact with the model, to change its angle of presentation. For example, it may be impossible to see all important features of a model from one viewpoint. Interactivity allows the user to rotate, translate, and scale the model so that he or she can view any features that are obscured in the initial presentation.

The requirement of model interactivity means that static images of a model embedded in a Design Web document will not suffice. An actual Alpha_1 viewing client must be directly embedded in the document. The technology that allows this is discussed in Section 2. This is only part of the solution. The viewer must still be modified to show an initial presentation of the model as specified by the designer. In order to satisfy this requirement, we implement *Model Views* for Alpha_1 viewing clients.

A model view is an encapsulation of the information necessary to provide a specific presentation of a model. This includes a model and all of its inherent information such as color, texture, and surface materials. It also includes view-specific information such as a camera location, scale, and information about which parts of the model are visible. Finally, a model view includes display properties that a viewer can apply to a model, but that are not inherent to the model itself. Each user might want to apply different display properties to a model. Some examples of display properties are visibility of surface normals, surface shading, and visibility of NURBS curve and surface control structures.

The Alpha_1 Implementation of Model Views

A model view consists of state information that is maintained within an Alpha_1 display library window object. Display properties such as surface shading, surface normals, etc., are window-specific and are therefore stored with each window via a lookup table. A window also maintains a list of which models it is displaying as well as the current camera position. Inherent properties of a model such as color and texture are also known, since these qualities are stored as attributes attached to the model graph. We notice that a model view is effectively an instance of a window object at a given time. The system supports model views by providing methods to read and write the state information stored by a window. In fact, the window itself is a first class persistent object that can be read and written via the Alpha_1 object stream mechanism.

Hyper-Links

A model view is essentially an "instance" of some portion of a design model. While this is sufficient to allow the designer to present a model within an Design Web document, it is not an entire solution. A design is typically represented hierarchically. The entire design is broken into components, each of which is recursively broken down further. In order to navigate this hierarchy, individual Design Web documents must be linked together. For example, an initial document might contain a version of the model with no detail, and only the main components. Each of the components would then be linked to individual pages which provide more detail for that component.

WWW documents are easy to link together. However, a model view is an illustration *embedded* within a WWW document and therefore must provide a mechanism to follow links to other model views or Design Web documents. Hyper-links are built into Alpha_1 in order to support this capability. The user is able to associate links with model objects using the Alpha_1 attribute mechanism. When a portion of a model that has a link associated with it is activated, the extension we are introducing in this paper loads either a new model view or an entirely new Design Web document as specified in the link.

Hyper-links are represented by existing Alpha_1 features called model attributes. Model attributes allow non-geometric information to be associated with any Alpha_1 model data structure. Custom Tcl/Tk code checks for hyper-link attributes on objects that are selected and takes the appropriate action (i.e., loading a new model view or jumping to a different web page.)

2. The Alpha_1 Web-Browser Plug-In

Browser Integration

An important step in realizing the Design Web is making Alpha_1 work with the other components. WWW technology appears to be an appropriate medium for integrating all the pieces of the Design Web. Other technologies may also meet the requirements, but we feel that the WWW is a good choice for some additional reasons. First, by using existing WWW browsers and plug-ins, one can leverage commercial off-the-shelf software. Although the technology is not *exactly* what the Design Web needs, it is close enough that the time saved by not re-inventing the wheel is a worthwhile trade-off. Second, it is possible to take advantage of the existing wide range of capabilities offered by the WWW. For example, a design that uses catalog parts might include a link to the company's web page so that up to date catalog information is readily available. Finally, the WWW is pervasive so distributing the Design Web to a wide user base is possible.

One of the interesting features of the Design Web concept is that live models can be mixed with other information in a WWW document. This capability could have been provided by implementing a WWW browser within Alpha_1. However, this is not an attractive approach since it would not allow the Design Web to take advantage of the pervasiveness of existing WWW browsers. A better method is to leverage existing WWW browser technology and embed Alpha_1 in a web browser.

The growth of the WWW is nothing short of explosive. Such a rapid increase in user base has spawned the development of a number of languages and other capabilities, such as the Netscape Plug-in Architecture, Java, and VRML, that are designed to ease the integration of user applications with web browsers. These technologies are new; there is no clear standard. Instead, one must look at each and use the one that is best suited to the requirements of the Design Web. Since Alpha_1 is a large system, one of the goals for integrating it with the web is to minimize the amount of code that must be rewritten.

Java

At first glance, Java appears to be an ideal choice for integrating Alpha_1 with the Web. Java is platform independent, integrated with virtually every available web browser, and can be automatically downloaded by the browser. However, we chose not to use Java in the current Design Web implementation for a number of reasons. First, Java is too slow. Although "just in time" compilers have improved the execution speed of Java applications, the performance improvements are not yet significant enough to make Java a viable alternative for computationally intensive applications. Second, the Abstract Window Toolkit (AWT), Java's GUI, lacks the power and flexibility that Alpha_1 programs have come to rely on. Finally, Alpha_1 viewers make heavy use of hardware accelerated 3D graphics engines such as OpenGL. Bindings for these libraries, such as Sun's Java-3D are only beginning to be available.

VRML

The Virtual Reality Modeling Language (VRML) has become a staple buzz-word in the world of the Web. Given its popularity, it must be considered as a potential medium for integrating Alpha_1 with the Web. On the surface, VRML has the look of a good candidate. However, we quickly discover that VRML is better suited as an Alpha_1 export format than a technology for hooking a large modeling system up to the Web. The major flaw of VRML is that it is essentially a scene graph specification and file format. While VRML 2.0 does allow for extending the format, the mechanism for accomplishing such extensions is not powerful enough to describe the complex behavior required by most modeling systems.

ActiveX

ActiveX is the main competitor to the Netscape plug-in architecture. It is appealing because of its COM heritage. ActiveX has the potential to allow Alpha_1 to be embedded not just in a Web browser, but in virtually any COM aware application. ActiveX was not chosen mostly because it is not yet available for UNIX, the primary operating system on which Alpha_1 is built.

Netscape Plug-in Architecture

The Netscape Plug-in Architecture was developed by Netscape Communications to allow the integration of their web browser, Netscape Navigator, with client applications. The Netscape Plug-in Architecture is an application programmer's interface (API), so the Alpha_1 C++ code can be accessed directly. It specifies a series of functions that the browser will call when a plug-in is to be initialized, viewed, and unloaded. The system suffers little performance penalty when running as a plug-in.

The Netscape Plug-in Architecture has a shortcoming in that plug-ins are created for a specific machine architecture and are therefore not portable. Although the Netscape Plug-in Architecture is not as portable an integration option as Java or VRML, it is used in the Design Web for the following significant reasons. First, the plug-in and the library or libraries associated with a plug-in are stored locally. This does not facilitate automatic updating when a newer version of the plug-in is available. However, the advantage to storing the plug-in locally rather than letting a browser cache it, as are Java applets, is that it only has to be downloaded once. This can be a big win for plug-ins that are large, as is the Alpha_1 plug-in. The performance of the Design Web would be unacceptable if the entire plug-in library had to be downloaded regularly if, for example, it got flushed from the browser cache like a Java applet. Second, the lack of portability is not much of a problem since Alpha_1 is already under concurrent development for four different operating systems (IRIX, HP-UX, Solaris, and Windows NT). Little extra work, if any, is required to create versions of a plug-in for each of these architectures. Third, plug-in code is native application code (C^{++} in the case of Alpha_1). Finally, the Netscape Plug-in Architecture is the best trade-off of ease of integration versus portability and level of integration.

The Alpha_1 Plug-in

The Netscape Plug-in architecture is the web integration method with the best trade-offs for the prototyping stages of the Design Web. Although this method does not require as much modification to Alpha_1 as any of the other integration methods, some changes need to be made in order to create a plug-in version of Alpha_1. The requirements for the Alpha_1 plug-in are that it must be flexible, expandable, and be designed with the future of the Design Web in mind. For example, as the Design Web is realized, the network integration technology might no longer be a Netscape plug-in. Therefore, the plug-in design avoids making use of any esoteric features of the plug-in architecture.

One of the biggest challenges of making an Alpha_1 plug-in is mixing the event models of the web browser and the Alpha_1 viewing clients. The Netscape Navigator[1], for example, utilizes the Motif widget set, which controls the application using its own set of widgets and event loop. Alpha_1 viewers, on the other hand, use the Tk toolkit [7], which requires its own widgets and event control loop. Even if Alpha_1 and Netscape used the same interface toolkits, the Alpha_1 plug-in would still need to be separated from the browser in case it ever needed to be used with a browser built on top of a different UI toolkit. Fortunately, a recent release of the Tk toolkit, developed to support a Tcl/Tk plug-in written by researchers at Sun Labs, allows Tk applications to be embedded within other applications that utilize different user interface toolkits. Using the Tcl/Tk

[1] The Netscape Navigator is likely to be the web browser that is used with the Design Web prototype most often. However, other browsers such as Microsoft Internet Explorer do support Netscape plug-in architecture, and might eventually be used with the Design Web.

plug-in from Sun as a model, the Alpha_1 plug-in takes advantage of this improvement to the Tk library. Netscape gives each plug-in a window within its own window hierarchy. This window must then become the top level window for the Tcl/Tk application that the plug-in manages. Note that each instance of the plug-in holds its own Tcl interpreter, thus preventing namespace conflicts that might otherwise arise from multiple instances of the plug-in attempting to load the same Tcl script. The use of IncrTk objects for encapsulation also helps to avoid Tcl namespace conflicts.

The other area where the toolkits used by the Netscape browser and the Alpha_1 plug-in must communicate is events. Each toolkit has its own event processing mechanism that it uses to respond to user input for toolkit widgets. Unfortunately, these toolkits are both designed for event driven applications, where the toolkit maintains control of the main thread and waits for the user to interact with one of the widgets on the screen. The UNIX versions of the Netscape browser use just such a toolkit, Motif. The main event loop of the browser application is not accessible to the plug-in, so it cannot merge in the Tcl/Tk event handlers. The only way to allow Tcl/Tk to respond reasonably well to user input is for the plug-in to get control from Netscape on a regular basis and handle any outstanding Tcl/Tk events that are waiting. This is accomplished by using the timer feature of the Motif toolkit.[2] The plug-in sets a Motif timer to execute a *callback* function regularly. The callback satisfies all outstanding Tcl/Tk events, up to some predefined maximum number, by calling a non-blocking version of the Tcl/Tk event handlers. The number of events that are handled at one time is limited, since it is possible for the user to generate new events as fast as the event handler can process them. If the maximum limit was not enforced, then the timer callback function might never return, or not return for an unreasonable amount of time. This would make the browser unresponsive to user input, even as the plug-in is happily processing user events.

The next issue to be addressed is the plug-in interface. The HTML EMBED tag is the primary mechanism for initial communication between Design Web documents and the Alpha_1 plug-in. The MIME types and additional arguments supported by the plug-in are of concern here. Alpha_1 provides two main functions for the extension we are describing. First, it provides modeling via the scl-eval parser function called from Tcl/Tk user interface scripts. Second, it provides interactive viewing via the a1_canvas Tk widget that is built on top of the Alpha_1 display library. The main goal of the current extension is to permit interactive viewing of a design and its history, so the Alpha_1 plug-in needs to support at least display of models and model views.

The plug-in architecture allows for a plug-in to translate multiple MIME types. The Alpha_1 plug-in takes advantage of this by mapping different plug-in functionality through different MIME types. This approach will be useful as the Design Web evolves and more functionality is expected of the plug-in. The use of multiple MIME types will allow a range of plug-ins that support different levels of integration with the Design Web to be available. For example, a "lightweight" plug-in might only support interactive 3D display of Alpha_1 models. Such a plug-in could be much smaller, and more accessible for the occasional user. This version of the plug-in supports only the model/x-a1 type. At the other end of the spectrum is the fully functional Alpha_1 modeling plug-in that supports viewing, interactive model creation, and anything else required by the Design Web. This plug-in would support all MIME types discussed here.

The primary MIME type used by documents in this extension is model/x-a1. This MIME type tells the plug-in to load Alpha_1 object files with the ".a1" extension into a viewing window. The object files are specified as SRC parameters in the embed tag and may contain arbitrary Alpha_1 objects, including model views. Another MIME type supported by the plug-in is model/x-a1-objid. Together with a distributed persistent object store, this type will allow the Alpha_1 unique object identifier to be specified by a parameter in the embed tag. The unique object id will allow the viewer to query a series of model servers in order to obtain a copy of the object with the requested identifier. Finally, the MIME type application/x-a1-edit, not yet implemented, will indicate that the plug-in should start a c_shape_edit server in the plug-in window to serve as a rendezvous for

[2] Most event-driven user interface tool kits provide some mechanism, such as a timer or an idle callback function, for giving control to an application on a regular basis. This event handling method will work with any tool kit that provides such functionality.

distributed design sessions. Users can then interactively execute modeling commands and send models to any display clients which connect to the server. This can include other Alpha_1 clients which are running as plug-ins.

Current Alpha_1 viewers and editors take advantage of the flexibility of the Tcl/Tk user interface toolkit. Tcl and Tk are extended with new commands and interface widgets that access the underlying Alpha_1 libraries. The main benefit of this approach is flexibility. The user-interface for Alpha_1 programs is controlled through scripts which can easily be changed to allow for end user customizations, experimentation, and rapid development of new interfaces. The plug-in was designed to work with the Tcl/Tk toolkit so that this power and flexibility is not lost just because the application is embedded within a web browser.

Currently, the plug-in allows customization of the user interface through the SCRIPT option to the embed tag. The user specifies the name of a Tcl script that describes the user interface to use for the plug-in instance. The name must refer to a script that is stored locally within the plug-in installation. If the script option is omitted, then the plug-in uses the standard viewer. The standard viewer, when run as a separate client program, is known as tk3d. An alternative interface which adds model construction and editing GUI controls for the two-view interface is known as shape3d. This approach was chosen for both convenience and safety. Since the scripts are referenced by name only, the user may customize these scripts locally (i.e., the named script can contain any Tcl code), and any document that references them will cause the plug-in to load and use the user-customized version of the script. This approach is somewhat safer than allowing scripts to be arbitrary URLs since the contents of the script can be any Tcl command, including file access commands. This is an acceptable limitation for the present extension. As the Design Web is developed, however, it may be desirable to allow the document server to specify the script along with the document itself.

Conclusion

Communications and computing technology to support aspects of distributed collaborative CAD is being developed by many different research organizations. Demand for the communications, computing and visualization products to support this kind of effort will become increasingly intense due to technological, economic, and life style pressures.

The work we have presented combines widely adopted web technology with an interactive interface to a design and process planning system. We have experimented with this system using a high quality video teleconferencing system to carry out full-scale distributed design and prototyping collaborations. The design team included design experts in two locations half a continent apart. The prototype system described in this paper represents a first step towards the next generation collaborative design environment.

Providing sophisticated, appropriate tools for designers in the field and across disciplines and distances is a challenging task that invites creative experimentation by researchers sensitive to the complexities of the disciplines involved. Our work is a contribution in this direction. This work and related web-based approaches are harbingers of a next generation of CAD systems that will allow us routinely to accomplish distributed, collaborative design.

As a specialized framework for *design*, the Design Web we are working on is a particular case of the larger area of computer supported collaborative work (CSCW). Although our work is concerned with specializations suited to CAD/CAM techniques, it can both draw from and contribute to advances in the larger field of remote collaboration. It is important to undertake, report, and compare such results to gain the understanding necessary for bringing about this new era of CSCW methodology.

REFERENCES

[1] CARLSON, D., 1997, A Web of Technologies, *Object Magazine*, 18-9.

[2] ENGINEERING GEOMETRY SYSTEMS, 1992. *Alpha_1 User's Manual*. Salt Lake City, UT.

[3] GUI, J.-K., and MANTYLA, M., 1994, Functional understanding of assembly modeling. *Computer-Aided Design* 26, 6.

[4] KRAUSE, F.-L., CIESLA, M., RIEGER, E., STEPHAN, M., and ULBRICH, A., 1995, Features – semantic objects for the integration of tasks in the product development process. In *Proceedings of the Computers in Engineering Conference and the Engineering Database Symposium*, ASME, ASME, 667-685.

[5] LOSS, B.F., 1997, DRIVE: A System for Interactive Review of Engineering Designs, Master’s thesis, Department of Computer Science, University of Utah, Salt Lake City, UT.

[6] OUSTERHOUT, J.K., 1994, *TCL and the TK Toolkit*. Addison-Wesley.

[7] STURGILL, M.M., COHEN, E., and RIESENFELD, R.F., 1995, Feature-based 3D sketching for early stage design. In *Proceedings of the Computers in Engineering Conference and the Engineering Database Symposium* , ASME, ASME, 545-552.

[8] TSAI, C.-S., CHANG, K.-H., and WANG, J.-Y., 1995, Integration infrastructure for a simulation-based design environment. In *Proceedings of the Computers in Engineering Conference and the Engineering Database Symposium*, ASME, ASME, pp. 9-20.

[9] ULLMAN, D., 1992, *The Mechanical Design Process*. McGraw-Hill, Inc.

[10] WANG, F.-C., RICHARDS, B., AND WRIGHT, P.K., 1995, A generic framework for the integration of mechanical and electrical cad tools for concurrent design of consumer electronic products. In *Proceedings of the Computers in Engineering Conference and the Engineering Database Symposium*, ASME, ASME, 131-146.

[11] WOYAK, S.A., MALONE, B., and MYKLEBUST, A., 1995, An architecture for creating engineering applications: The dynamic integration system. In *Proceedings of the Computers in Engineering Conference and the Engineering Database Symposium*, ASME, ASME, 1-8.

[12] ZHANG, S., HOU, X., and WANG, Z., 1995, Step application protocol interoperability for cad/capp integration. In *Proceedings of the Computers in Engineering Conference and the Engineering Database Symposium*, ASME, ASME, 687-690.