DRIVE: A SYSTEM FOR INTERACTIVE REVIEW OF ENGINEERING DESIGNS

by

Brian Frank Loss

A thesis submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science

The University of Utah

December 1997

THE UNIVERSITY OF UTAH GRADUATE SCHOOL

SUPERVISORY COMMITTEE APPROVAL

of a thesis submitted by

Brian Frank Loss

This thesis has been read by each member of the following supervisory committee and by majority vote has been found to be satisfactory.

Chair: Richard F. Riesenfeld

Elaine Cohen

Peter Shirley

THE UNIVERSITY OF UTAH GRADUATE SCHOOL

FINAL READING APPROVAL

To the Graduate Council of the University of Utah:

I have read the thesis of Brian Frank Loss in its final form and have found that (1) its format, citations, and bibliographic style are consistent and acceptable; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the Supervisory Committee and is ready for submission to The Graduate School.

Date

Richard F. Riesenfeld
Chair, Supervisory Committee

Approved for the Major Department

Robert R. Kessler
Chair/Dean

Approved for the Graduate Council

Ann W. Hart
Dean of The Graduate School

ABSTRACT

Computer aided design and manufacturing tools have allowed designers to enjoy significant reductions in the design cycle. A shorter design cycle is advantageous for a number of reasons, including shorter time to market, reduced development costs, and superior product design. Current CAD/CAM tools are mature and provide comprehensive support for many individual stages of the design process. A challenge lies in creating systems with fluid support for all stages of the design process, and support for the design process itself.

This thesis describes Design Review Interactive Viewing Environment, a first step towards the goal of design process support. The Alpha_1 modeling system is integrated with the World Wide Web, yielding an effective tool for interactive design review. The design process is captured in a hypertext document that integrates nongeometric information such as design rationale with interactive CAD models. Existing web browsers make it straightforward to include design descriptions, sketches, pictures, and so on in a hierarchical document structure. A web browser plug-in allows Alpha_1 model viewing and editing clients to be embedded directly in hypertext documents. In addition, Alpha_1 is extended with model views, which are specific presentations of a model. Hyper-links provide a mechanism for model views to be linked to one another or to other web pages, thus allowing navigation of a design hierarchy from within Alpha_1 clients or their plug-in counterparts.

CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

This work was supported in part by DARPA (F33615-96-C-5621) and the NSF Science and Technology Center for Computer Graphics and Scientific Visualization (ASC-89-20219). All opinions, findings, conclusions or recommendations expressed in this document are those of the author and do not necessarily reflect the views of the sponsoring agencies.

CHAPTER 1

INTRODUCTION

Computer aided design and computer aided manufacturing tools have revolutionized mechanical design. The use of a CAD package during the later stages of a product design is practically required if the designer is to succeed. However, integrated CAD support for the early stages of the design process, as well as support for the process itself is lacking [12]. A system capable of supporting the entire design process in addition to individual stages of design could provide significant further reductions in the product design cycle.

This research provides a first step towards such a system. In particular, it focuses on information integration through support for the design process itself. A finished design has much more information associated with it than just a CAD model. A design goes through an evolution, often before any information is entered into a CAD program. There are functional specifications, alternate design choices, and so on. Furthermore, designs are often hierarchical in addition to iterative. That is, a large design is recursively broken into smaller pieces. Each of these subdesigns goes through a similar evolution as well.

This thesis develops a prototype system, integrated with the Alpha_1 Geometric Modeling and Manufacturing Software System [3], that allows the user to define hierarchical "views" of a model created using Alpha_1. These "views" can then be linked together in much the same way that World Wide Web documents are linked. This furnishes the user with a convenient way to navigate through a design, as well as focus on a manageable piece of a large or detailed model. The prototype system also makes use of recent technology, thus allowing Alpha_1 viewing clients to be embedded within the Netscape Navigator, a popular WWW browser program.[1] An advantage of this integration is that each view of a model can be associated with nongeometric design information, thereby furnishing the designer with a method to keep records of the design process itself.

1.1 Motivation

Due to the increasingly competitive marketplace, design engineers face the challenge of providing high quality designs in continually shorter periods of time. They are further burdened by the task of supporting a wide variability of any given design. Today's consumers have come to expect manufacturers to be able to meet their specialized needs in a short period of time and for a relatively low cost. [2, 9]

A shorter design cycle helps a company gain a competitive edge for a number of reasons.

Reduced development costs: Good designers are a valuable commodity, and any reduction in the design cycle for a product means that the company is either not paying for as much of the designers' time or that the designers are available to work on other products.

Better timing of product releases: Often, the timing of a product's release is a major factor in determining its success. The tastes of the marketplace can change rapidly, and a product might be "out of style" if a product is delayed because of problems in the design stage. Also, product releases are often event driven. Christmas will not be delayed just because a company cannot get its product out in time for the holiday shopping season. A shorter design cycle means that designers are able to work with much more accurate assessments of the market for their product. They can then tailor the product to meet the current preferences of their customers. With a longer design cycle, it is more likely that those needs will change at a point when the product design can no

[1] The browser plug-in can be used with any browser that supports the Netscape plug-in architecture, so long as Alpha_1 is able to run on the operating system required by the browser.

longer be altered.

Reduced capital cost: Product development requires a substantial capital outlay, an investment that provides no return while the product is still in the development and prototyping stages. Thus, there is a compelling financial reason for introducing the product into the market as soon as possible. This is analogous to the building contractor who starts renting completed floors of a building even though other floors are still under construction.

Superior product design: A shorter cycle allows designers to make more modifications and improvements on the original design with no cost increase. The designers can carry out more iterations toward the final design, and consider more design alternatives.

Smaller lot sizes: Consumers have specialized needs. It is impossible for manufacturers to satisfy everyone, but the decreased cost associated with the shortened design cycle can make it cost effective for companies to produce smaller runs of any given product. The ability to produce small numbers of a wider variety of products can be advantageous for both the producer and consumer. Smaller lot sizes may also be beneficial from a marketing standpoint. Auto makers, among others, have discovered that the words "limited edition" can often increase the demand for a product.

CAD tools have provided design engineers with the means to significantly shorten the design cycle. Design is an iterative process that often forces a considerable amount of repetitive work [12]. CAD tools allow the designer to take advantage of this by permitting simple modifications of an existing model and virtually eliminating the error prone process of copying a previous design. Current progress in many areas of CAD notwithstanding, significant problems still need to be solved.

Even though current CAD tools give the designer considerable flexibility, they still do not support the entire design process. In particular, adequate support for the earlier stages of design is lacking. Although there is some support for the earlier stages [4, 10], it is often divorced from downstream processes. For example, early stage design tools such as sketching tools do not produce the detailed three-dimensional models that current CAD systems require as their input. Such models are usually not available until later stages of the design process. Since the early stages make up a large portion of the whole process [12], designers stand to gain considerably from improved computer support in these stages.

Current CAD tools have become highly specialized, providing designers with a wide array of capabilities from computer aided drafting to realistic rendering to finite element analysis. Once again, there is a difficulty in that each of these programs requires a highly specific type of input. This input is often incompatible with that required by other programs. The end result is that no system provides comprehensive and compatible computer support for *all* phases of design. Even if tools that meet the designers needs are available, they are likely to be separate packages or separate programs within one package.

This lack of *integration* is a serious deficiency found in current systems. It can make the different programs tedious and inefficient to use, as much of the user's time is spent switching from program to program or converting a model among different formats. In general, it implies a lack of support for design as a process. The individual stages of the design process are not as important as the whole process. Although structural analysis, for example, can certainly be a critical component of a design, the effects of this analysis on the rest of the design are even more important. If the structural analysis indicates that a change needs to be made, the user should be able to make the change and have the CAD system automatically propagate it as necessary.

Fluid interaction among different modules of the system is even more important during the earlier stages of design, when many important design choices are made. The system should allow for tracking and merging of alternatives that result from either concurrent design or different iterations of the design cycle. Ideally, the system should also be able to apply tools that are normally intended for later stages of design as early in the process as possible. Even if these tools are only able to provide a rough estimate (because the model is not fully specified at this point), they can provide crucial information to the designer. For example, designers working on an airplane would like to know as soon as possible if their component of the design has exceeded a budgeted weight. If a rough estimate of the final weight shows that the limit has been drastically exceeded, then the design needs to be reworked. Clearly, this knowledge can save the designer a great deal of time if it is available early in the process.

The next generation of CAD/CAM tools will begin to address these problems. Now that many single-stage tools have matured, researchers are beginning to concentrate on making the technologies work together [6, 11, 13, 14, 16]. Designers will have fluid access to many different design tools such as renderers, analysis packages, and parts databases. The system will provide early stage design support—perhaps a sketching program that allows the designer to sketch out a rough idea and then fill in the corresponding details in the CAD model. At any point in this process, the model will be fluidly usable by virtually any part of the system. In addition, we will start to see more support for the design process itself. This will include "process oriented" capabilities such as design alternative tracking.

Providing support for the design process uncovers a new set of problems. To some extent, supporting the design process means unrolling the design loop. The result is that an enormous amount of information can be associated even with relatively small designs. With access to so much power and flexibility, the user can easily become overloaded with information. The system needs to provide a way for the user to deal with the complexity of a large design as well as the complexity of a large CAD package. If the user cannot readily access this "process" information, an integrated design system will lose many of its advantages.

1.2 DRIVE Overview

The *Design Review Interactive Viewing Environment* DRIVE provides the framework of an effective environment for distributed engineering design review. It also provides the basic building blocks needed by *The Engineering Design Web*, an integrated design environment consisting of "live" design documents. EDW documents integrate text, graphics, Alpha_1 modeling clients, and other media in a WWW-like document that can be the basis for a distributed collaborative design session. A more detailed description of the EDW concept is provided in Chapter 3.

DRIVE provides two main additions to the Alpha_1 modeling system: *Model Views* and an *Alpha_1 Web Browser Plug-in.* A model view contains information about how to present a model to the user—what parts of the model are visible, a view point, a view scale factor, display properties (shaded surfaces, surface normals, etc.), and any other viewing information that is window-specific. Essentially, a model view is an active model viewing client with persistent viewing state. This state information is saved in the same way as other Alpha_1 model objects.

Model views can also be *hyper-linked* in much the same way as HTML documents. Selecting part of a model in a model view causes another model view to be loaded, a URL to be loaded in a running web browser, or both. The user can take advantage of this feature by linking together different models from a design. For example, a model view with rough geometry can be linked to other model views that contain more detailed geometry. Those detailed models could in turn be linked to even more detailed models, an so on. Such a hierarchy of linked model views allows the user to effectively navigate, or browse, the design space simply by clicking on parts of interest.

DRIVE also adds a web browser plug-in to Alpha_1. Based on the Netscape plug-in architecture, the browser plug-in allows most Alpha_1 programs that utilize the Tcl/Tk [8] user interface toolkit to be embedded in HTML documents. When a browser supporting the plug-in architecture displays such a page, it activates the plug-in and provides the user with an active Alpha_1 client that is effectively a part of the page.

DRIVE represents a first step towards creating the next generation CAD system described above. Alpha_1 is particularly well suited for this task because of its unique implementation. It is a client-server model with a dependency graph representation of the model. By taking advantage of Alpha_1ś ability to store a persistent version of the model dependency information, DRIVE adds tools to provide hierarchical "views" of a model and allows the user to navigate among these views and their associated design information. In short, DRIVE brings some "web" functionality to Alpha_1 through model views, and incorporates Alpha_1 into the web via the web browser plug-in.

CHAPTER 2

BACKGROUND

2.1 Alpha_1

Developed at the University of Utah, Alpha_1 is a parametric CAD system with integrated CAM support [3]. The system stores geometry using a trimmed NURBS boundary representation, thus allowing full three-dimensional modeling. Software support allows the user to generate and simulate corresponding code for numerically controlled machine tools. The system also supports nongeometric data in the form of model attributes that are stored with the model, and quasigeometric objects such as block diagram boxes and arrows.

Alpha_1 is designed according to a client-server architecture (Figure 2.1) where model geometry is held by a server in a dependency graph representation. The dependency graph representation allows parametric modeling. Most models are created using the shape_edit command language (SCL), a modeling language that is parsed into dependency graph objects and then interpreted by dependency propagation. The dependency propagation causes constructor functions, which generate

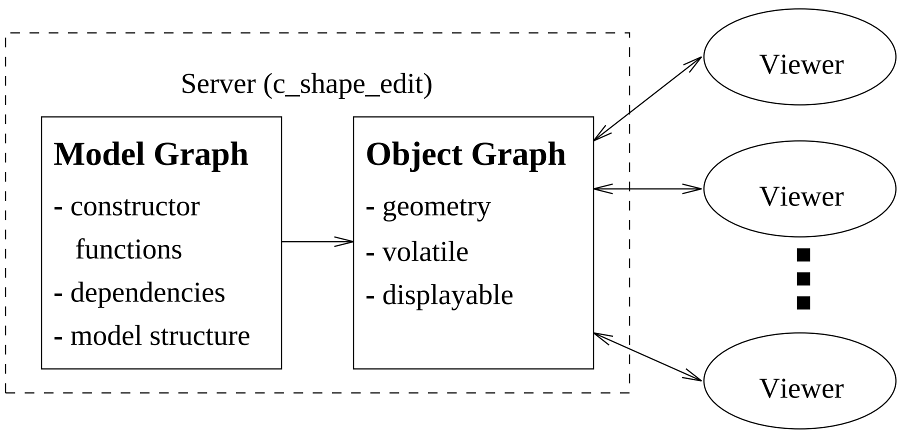

Figure 2.1. The Alpha_1 client-server architecture.

the actual model geometry, to execute on demand. Other client programs such as model viewers obtain constructed geometry from a `c_shape_edit` server. The server notifies the connected programs when the model changes, then they can request updated geometry from the server if necessary. Furthermore, all Alpha_1 programs can read and write binary "persistent object" versions of a model. In fact, the server sends models to clients by writing them in binary form on a socket connection.

Although much Alpha_1 modeling has been done using a text-based method for specifying geometry, recent work (e.g., [10]) has begun to add support for graphical geometry specification. Graphical user interface clients construct objects and send them, via the server, to other Alpha_1 clients. The next section describes some unique capabilities of Alpha_1 that allow a graphical interface for model design to work well in conjunction with the text-based interface.

2.1.1 The Two-View Interface

The client-server architecture of Alpha_1 leads to a highly modular system with a central representation of the model. This organization provides some interesting capabilities. In particular, it is possible for the system to generate SCL code from the internal representation of a model. This yields a two way connection between the graphical and textual representations of the model (Figure 2.2). The "two-view interface," first suggested by Banks [1], thus gives the benefits of both text-based

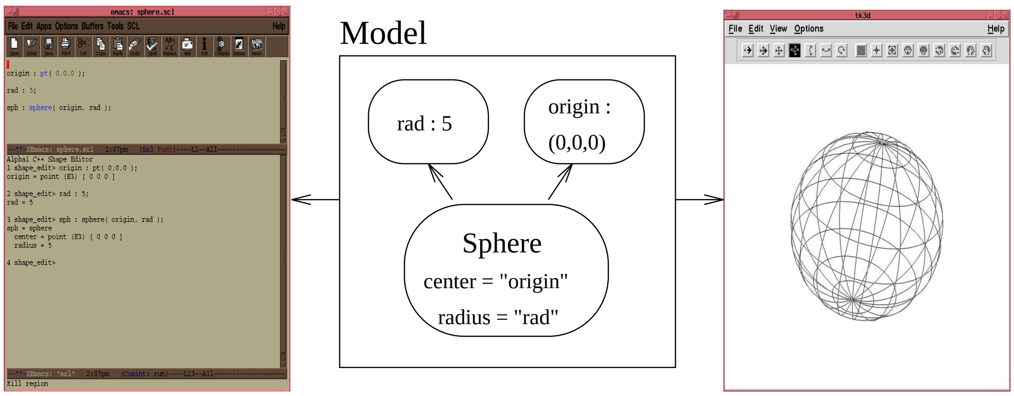

Figure 2.2. Textual and graphical views of a model.

and graphical interfaces without taking away any power from either. Operations that make more sense to perform graphically, such as sketching and initial geometry specification can be performed that way. The same model can be modified through the SCL language when exact values for some dimensions are known, for example.

2.2 Related Work

DRIVE involves a number of different research areas. Specifically the system draws on previous work and generates new ideas for design process support, collaborative design, and WWW integration. A wide variety of research is available in these areas, especially collaboration and WWW integration. However, relatively little of this work involves the integration of these different technologies to the extent that DRIVE, and the EDW accomplish. We suspect that a number of other researchers are working on similar problems, especially those involving the integration of full strength applications with the WWW. As these areas continue to grow, a much wider base of related work is expected to be available.

2.2.1 Design Process Support

Surprisingly little work has focused on supporting the entire design process and providing tools to make the associated information available through one convenient interface. Many researchers have concentrated on providing better CAD support for various individual stages of design, but few seem to concentrate on linking support for these stages together. For example, a number of reports [5, 10, 15] describe systems to aid the early stages of design, when the designer tends to make rough sketches. Most notably, the sketching tools described by Sturgill et al. [10] are directly integrated with a CAD system. In general, however, these tools do not take the rest of the design cycle into account.

The work presented by Tsai et al. [11] represents significant progress in integrating later stages of design in a simulation-based design environment. The authors use the interesting approach of presenting different views of a model based on discipline, such as reliability, maintainability, and so on. Unfortunately, the system described requires a complete CAD model. In addition, although the authors discuss some "process oriented" abilities such as tradeoff analysis, it is unclear if their system provides any kind of history or record of the design process. It appears that this system is useful for driving later stages of a design, and iterations of the design. However, much of the design process takes place before the first CAD model is created. This significant portion of the design process must be recorded along with the later stages.

The Dynamic Integration System [14] is a method for integrating and coordinating modules of engineering software. Modules make their needs and functionality known, and the Dynamic Integration System uses special data structures, called Dynamic Variables, and a dependency hierarchy to integrate various modules. This intriguing concept is somewhat akin to a dataflow system, and is not well suited for the integration of some nongeometric design information such as specifications and design rationale.

Krause et al. [6] present a work that begins to address the need for computer support for the entire design process. Their system is based on "features," which in the context of the authors' work are semantic objects that can carry information related to the design process. The system treats features much like any other ordinary design object, thus facilitating their integration in the design process.

Wang et al. [13] address the lack of communication between designers working in different disciplines on the same product. Many electronic products are eletro-mechanical, and require electrical as well as mechanical design. Typically, designers in the two disciplines tend to ignore each other until very late in the design process. The authors present a work that addresses some of the issues involved with multi-disciplinary product design. Data sharing, the main hurdle to overcome in a multi-disciplinary system, is accomplished through a scheme the authors refer to as "Distributed Relevant Data Exchange." Rather than require a common file format or a common data location, data for each group of CAD tools in a given domain maintain is stored in a common database. Some information is deemed to be relevant to other domains, and is shared with them via a global database manager. This type of data management could be useful for extending the design web to handle multiple design disciplines.

2.2.2 Collaboration

Computer Supported Cooperative Work, or collaborative computer tools have seen a recent surge in popularity. This area furnishes a great deal of research for the EDW to draw on as it develops. DRIVE borrows some ideas from this area in order to provide tools that will allow the EDW to be an effective collaborative design environment.

One of the most common collaborative software tools available today is the shared whiteboard. This is typically an enhanced paint program that allows multiple users to "share" a canvas. The participants can add text, draw figures, and paste images on the canvas as well as see and modify the object placed on the canvas by other users. Such tools are often incorporated with audio and/or video conferencing tools, other common collaborative tools. The InPerson software from Silicon Graphics, Inc. is a typical example of such an application. While quite useful in their own right, these tools run out of power quickly when used for collaborative modeling. The lack of a full CAD model, or even three dimensional geometry is the crippling factor.

DRIVE makes a first pass at supporting collaborative design by allowing multiple participants to view a design over the WWW. The "shared" model can be changed from a single server location (using the existing client-server capabilities of Alpha_1) and the change propagated to all viewers. Although many other applications allow shared display of three-dimensional geometry (e.g., [7]), none that the author is aware of integrate the viewing client with a complete CAD system as the DRIVE system does. The most common form of distributed viewing supported by current CAD systems is an option to export to VRML, a three-dimensional model file format supported by most WWW browsers (or plug-ins for those browsers). While DRIVE currently adds little or no functionality over that provided by these systems, we feel the CAD system integration provided by DRIVE will allow it to quickly outpace competing systems as development continues.

2.2.3 WWW Integration

With the explosive growth of the WWW, the desire to make it more interactive has expanded. Browser plug-ins, ActiveX, and Java are all results of the huge efforts to allow the WWW to be more than just a medium for viewing static HTML documents. Although these technologies are barely one year old, the number of ActiveX controls and Netscape plug-ins available is already quite large and growing rapidly every day.

Despite the fact that WWW integration is such a popular idea, relatively few plugins have encapsulated existing full applications. Most available plug-ins that we have seen are lightweight viewers of some kind, usually with limited functionality. One example of this is VRML viewers such as the Live3D viewer distributed with the Netscape Navigator WWW browser or the Cosmo player from Silicon Graphics, Inc.

One reason that plug-ins are generally not encapsulating large scale applications is the limited bandwidth of the WWW. Even users with a fast connection will not tolerate waiting for multi-megabyte applications to download with their web pages. Also, much of the web integration technology is quite new and developers have either not had a chance to integrate existing applications, or are not able to because the integration strategy will not support it. Java, for example, is a new language with limited support for making native calls to C or C++ methods. This makes it difficult for a developer to write a Java version of an existing application without completely re-writing it.

DRIVE avoids these issues by utilizing the Netscape plug-in architecture for the Alpha_1 browser plug-in. In short, the plug-in architecture provides hooks for plug-in functionality, so integrating an existing code base is relatively simple. The bandwidth is not much of an issue because the plug-in must be stored locally rather than downloaded on demand when a page requiring it is loaded. Although less flexible than the run-time download options provided by other languages such as Java or ActiveX, we feel this limitation is acceptable given the intended use of

DRIVE and the EDW. The Netscape plug-in architecture and its use with DRIVE are explored in Section 4.4.

CHAPTER 3

THE ENGINEERING DESIGN WEB

3.1 Overview

The Engineering Design Web is, in part, an extension of the two-view concept described in Section 2.1.1. It is also one of the future directions for the Alpha_1 system. Providing more powerful modeling capabilities, the two-view interface integrates the textual and graphical representations of a model. Although the model is certainly a critical part of a design, it is not the entire design. There are many other equally important aspects of a design, including:

- functional specifications
- block diagrams relating functions
- mapping of functions to parts
- part assembly interfaces
- initial sketches
- design alternatives
- design analysis information

Computer support for these design components does exist. However, by supporting just one or two design areas, current packages force specific design process boundaries on the designer. The designer must explicitly move data among the different tools. A long-term goal of the EDW is to remove these artificial design process boundaries, making the tools more transparent and freeing the designer to concentrate more on the design.

One starting point for achieving this integration is to encapsulate all of the design information mentioned above in a "live," hierarchical design document whose structure looks much like a formatted HTML document (Figure 3.1). Information such as specifications, design notes, or related formulae and calculations could be mixed with graphical and textual displays of a model. As with HTML documents, this EDW document would contain hyper-links that point to other EDW documents (a more detailed specification and model of part of the entire design, for example) or other sources of information such as a parts catalog. The document is "live" in that embedded models can be fully interactive, as opposed to static pictures of models. A live model can be manipulated using whatever mechanism the document plug-in

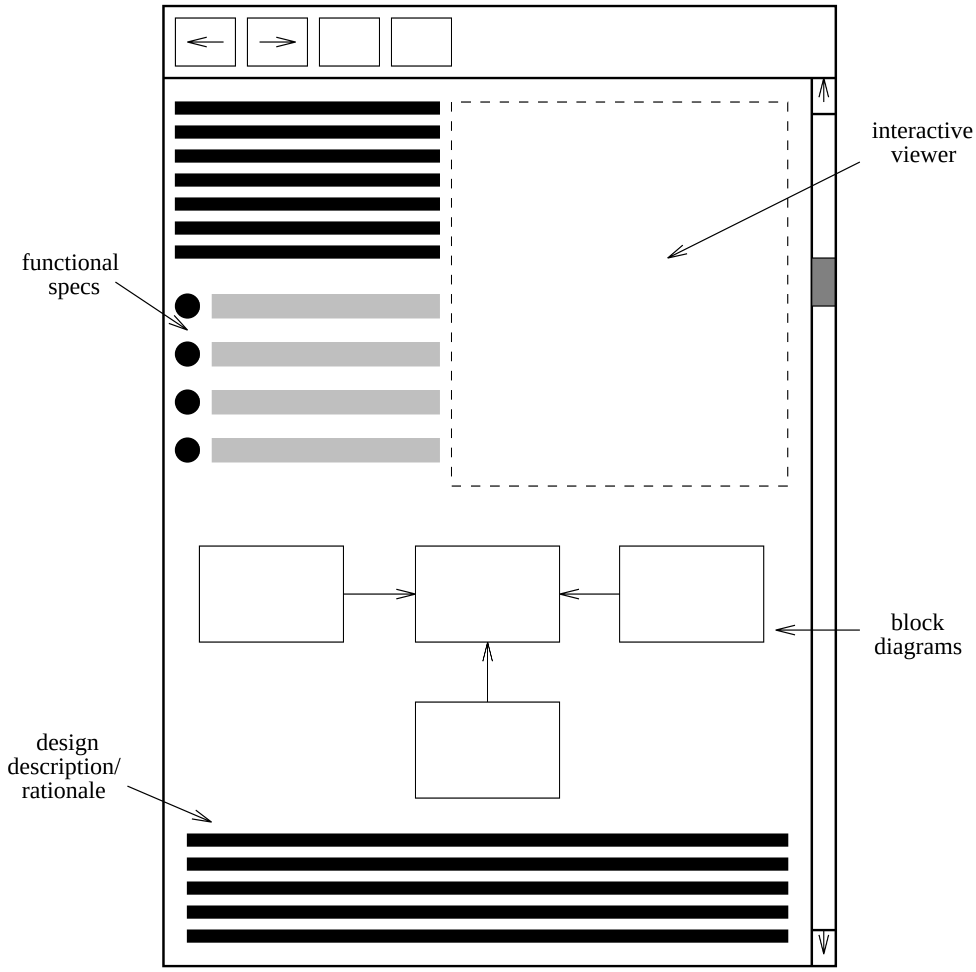

Figure 3.1. A prototype EDW document.

provides. This might be just view manipulation, or maybe full editing capabilities provided by an underlying CAD system.

DRIVE represents a first step in this new direction for CAD systems. DRIVE implements enough of the EDW as described above to allow Alpha_1 models to be embedded in HTML documents. The models are viewed or edited from within the document using plug-in versions of standard Alpha_1 modeling clients. Furthermore, the models themselves can contain links that point to other models or HTML documents. A number of approaches for achieving this integration were considered. They are discussed in Chapter 4 along with the DRIVE infrastructure.

3.2 DRIVE Example

This section presents a brief example of the EDW concept, as implemented by DRIVE. The following images show a real design of the spindle for a numerically controlled milling machine.[1] The design has been documented using the DRIVE system.

Figure 3.2 shows the top level EDW document, which looks no different from most other HTML documents at first glance. The document contains text describing the high level approach to the spindle design, links to background information and terms that some readers might not understand, and a rough picture of the NC machine's main moving parts.

When the user moves the pointer into the picture, we discover that the picture is not just a picture (Figure 3.3). The picture is in fact, an instance of the Alpha_1 plug-in. Moving the pointer into the plug-in window causes the model viewer user interface to appear. The user can use all of the features that are in the stand alone version of the viewer. Figure 3.4 shows the same window after the user has rotated the model to a different viewpoint.

The initial view in the plug-in window is an example of a model view. The model view contains information to indicate the the model is displayed using shaded surfaces, that one portion of the model is selected (shown in red in Figure 3.3). The

[1]Spindle design data courtesy of Bridgeport.

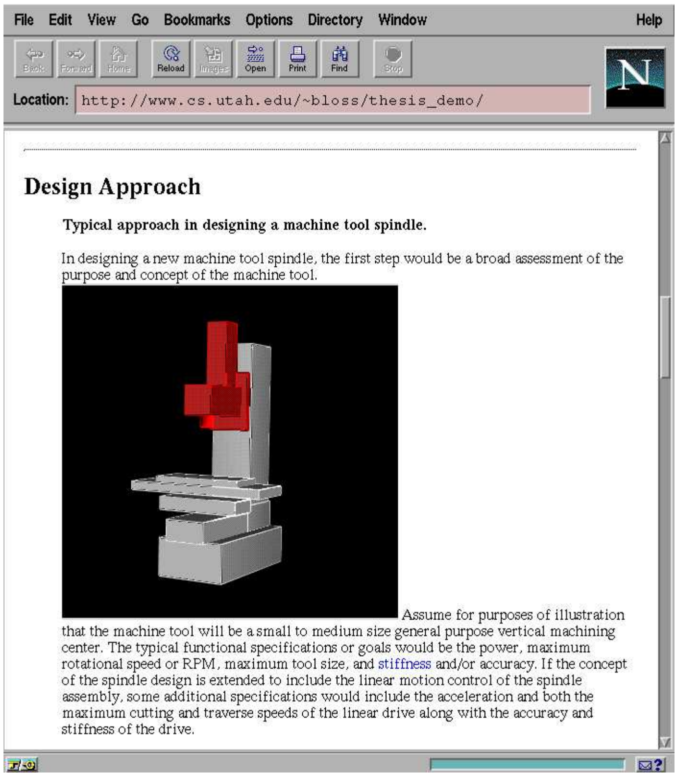

Figure 3.2. An example EDW document.

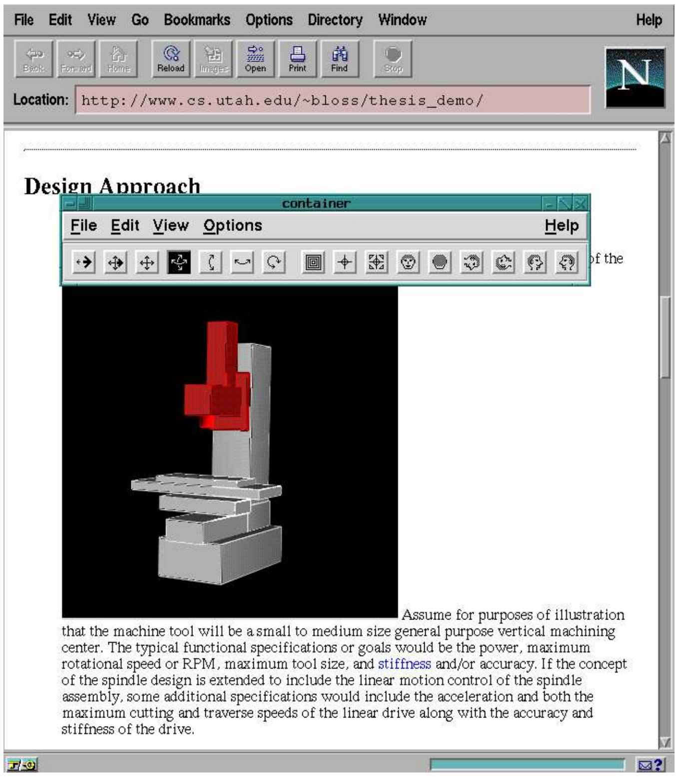

Figure 3.3. Plug-in viewer control appears when the pointer is in the plug-in picture.

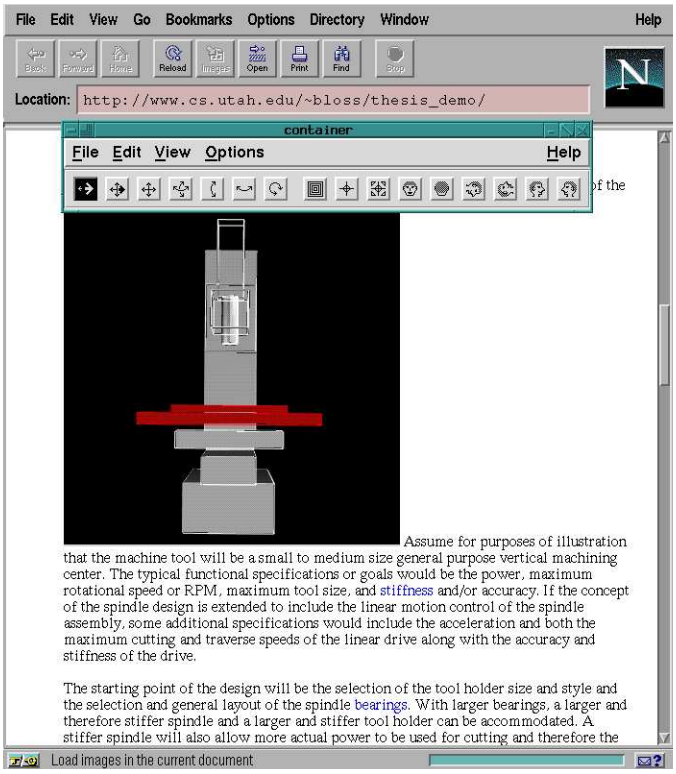

Figure 3.4. The plug-in picture is a full Alpha_1 viewer. Here, the model is viewed from a different vantage point.

viewpoint is also stored with the model view. Reloading the EDW document will cause the default view associated with the document to be reloaded by the plug-in.

Another feature of model views is the ability to add links to other model views or EDW documents. The selected portion of the model shown in Figure 3.3 has a link to a more detailed model of that portion of the NC machine. In this case, the more detailed view is a model of the spindle. The user clicks on the selected portion of the model to load the new model view, shown in Figure 3.5.

A single document can contain multiple plug-in instances, if multiple models or multiple views of the same model are to be shown. Scrolling through the example EDW document reveals a second plug-in instance (Figure 3.6). This plug-in also contains a view of the spindle model. The surrounding text describes the model in some detail. This text, and the model view associated with it could have been placed in a separate EDW document. For larger designs, or even a more complete description of the spindle design, this would most likely be the case.

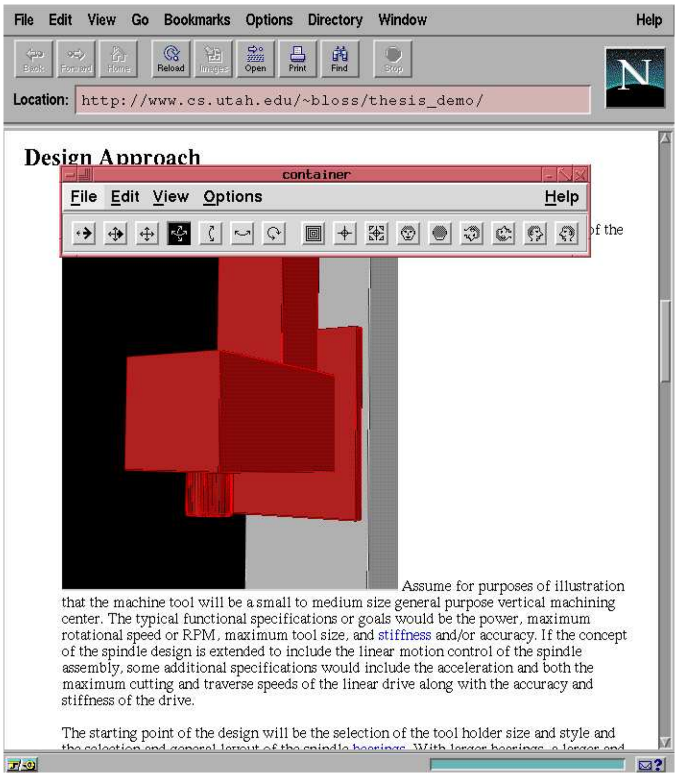

Figure 3.5. Model views can have links to other model views.

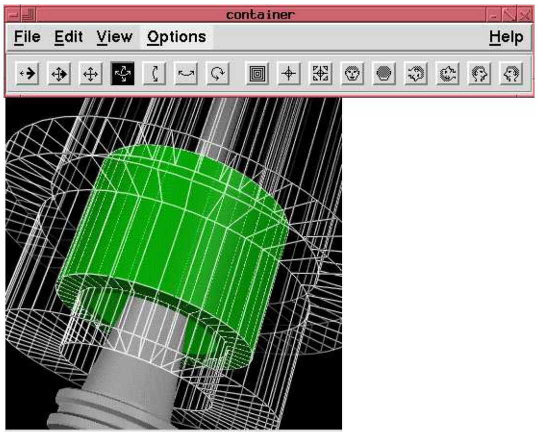

Figure 3.6. Model views can have links to other EDW documents.

The model view shown in Figure 3.6 has a link to another EDW document. In this case, the document is an HTML page with a single plug-in instance (Figure 3.7). This instance executes the `shape3d` Tcl script, which implements a model editor. EDW documents can use this mechanism to allow the user to edit models.

3.3 Future Work

DRIVE is a first generation prototype, and a number of areas are open for continued work on the concept. Although DRIVE allows the designer to record some of a design history, it needs some improvements. First, the design document is not automatically generated. It would be useful to remove the burden of setting up model views, setting up the HTML documents with the Alpha_1 plug-in, and so on from the designer. Any automation that can be introduced means less time that the designer will have to spend on it. This has obvious advantages in both decreased design time and the likelihood of the designer creating a more complete document.

Second, the ability to track design alternatives as well as just a history is another desirable feature. Often during a design many alternatives are explored. The final design may result from choosing one path, or combining the strengths of many alternatives. The EDW is a good place to record this information. Another area for continued work on the EDW is the development of the prototype into a full-fledged design system. The Alpha_1 plug-in is a good first step. However, the plug-in and

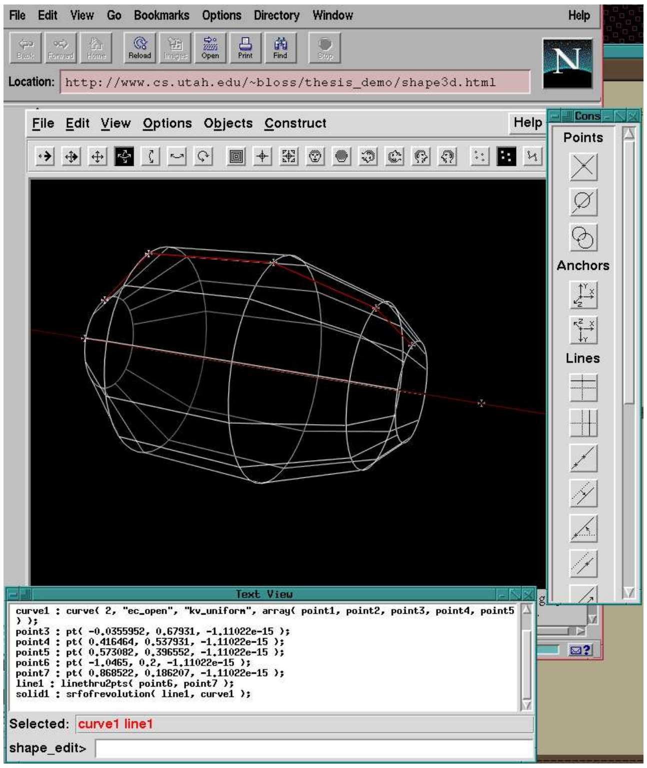

Figure 3.7. Virtually any Alpha_1 Tcl/Tk script may be run as a plug-in. This is the `shape3d` client, run as a plug-in.

the integration of Alpha_1 with the web need to be advanced to the point where the EDW can be an effective design environment as well as a design review tool. This includes additions to Alpha_1 such as the design alternative tracking mentioned above. It also includes the addition of distributed design capabilities that would allow designers at separate locations to collaborate on a design.

Third, the Alpha_1 plug-in needs to evolve along with the EDW. That is, the prototype plug-in provided by DRIVE must be enhanced to meet the needs of the EDW as it grows. In particular, support for the MIME types described in this thesis but not implemented for DRIVE need to be completed. Another problem with the plug-in is its size. Alpha_1 is a large system, and a significant portion of it must be included in the plug-in. Even though the overall size of the library or libraries may be relatively fixed, the cost of initializing the plug-in can be reduced. Currently, each instance of the plug-in maintains its own Tcl interpreter. Thus, when an instance of a plug-in is initialized, a new interpreter is created and all of the Alpha_1 extensions to Tcl are added to it. One potential method for reducing the cost of initializing a plug-in instance is to modify the plug-in so that all instances share a single interpreter.

Fourth, DRIVE will require some form of security if it is to be used in the EDW. An EDW document may contain proprietary information that the designers do not want to allow just anyone to access. Some people may be allowed access only to specific sections of a design. Others may be allowed to look at the design documents but not edit them or the embedded models. If the EDW is ever to become a general tool that sees widespread use, then these issues will become critical. Existing security implemented by most HTTP servers should prove useful, but it is certainly not an entire solution.

Finally, the EDW will need its own server. DRIVE relies heavily on existing WWW server technology to provide its documents. This is acceptable for the first pass, but the limitations of the existing technology will quickly be reached. The next step is to combine the Alpha_1 `c_shape_edit` server with an existing HTTP server, such as Apache. A combined server enables a much more flexible overall system. EDW document text can be stored with models (either directly with the model, or associated with the model in the same manner as display properties) and then used to generate an appropriate HTML document when it is requested from the server. In this scenario, the designer creates all the components of the document using potentially many different tools. The server puts the components together into a final product when the document is requested. This assembly can differ based on who is requesting the document. For example, the server can take user preferences into account and generate the document based on each user's needs. The server can exclude information to which the requesting party is not allowed access.

Perhaps the most useful feature made possible by the combined server idea is that the document can be served in a form appropriate to the level of technology available to the requester. That is, the server can generate a document that contains only images of the model if the user making the request does not have the Alpha_1 plug-in installed. Or, perhaps a VRML model could be substituted for an Alpha_1 model is the browser has a VRML viewer but not the Alpha_1 plug-in.

CHAPTER 4

THE DRIVE IMPLEMENTATION

4.1 Overview

One of the main purposes of DRIVE is to capture more of the design process than current CAD systems. Capabilities of most systems include the ability to work well with geometry, relationships/constraints, features, and text attributes. This research concentrates on two aspects of the design process that are not handled well, or at all, by current systems:

1. nongeometric design information,
2. design process history.

Nongeometric design information can include design specifications, parts catalogs, model analysis data from techniques such as finite element analysis, and design rationale. All of this information is essential to the successful completion of a design. Designers can benefit from having this information readily available through the CAD system itself rather than from other sources such as paper notes. Such a system will make it easier for the designer to record nongeometric design information, thus giving the designer more time to provide a more detailed design rationale.

Nongeometric design information is closely tied to the design, and to the eventual CAD model that results. However, this information is needed at various points during the design process. For example, a parts catalog may be used to select some standard parts, such as bearings or screws, for a model. However, the exact specifications of the parts need not be known until fairly late in the process. It would be counter-productive to require the designer to specify too much information during early stages of the design, when the designer often does not have all the details. Furthermore, a system that requires the designer to enter too much information will interfere with the conceptualization of the design. Functional specifications, on the other hand, come into play right from the start. Many requirements *must* be clarified before the design can progress at all. Since nongeometric information is so pervasive throughout the design process, it is not sufficient to provide the designer with a separate tool to record it. Ideally, the system should not require or even ask for any information before it is necessary in the design process. The EDW must take this into account by allowing nongeometric information to be fully integrated and readily available at any time.

Design is an iterative process. Virtually every significant design goes through an evolution before it is complete. Specifications change, analysis shows that some specifications have not been met, catalog parts are not always available and customer needs change. This means that a design undergoes many changes before it is complete. Some parts may be redesigned completely. Other parts may be designed by multiple teams, and the best approach (or maybe a combination of multiple approaches) is used. The history of a design is a hierarchy that the EDW must encapsulate.

DRIVE satisfies these requirements for the portions of the EDW it implements by integrating existing Alpha_1 clients with WWW browsers and hyper-linked HTML documents. The idea of a live hyper-linked document matches up well with the above requirements. The integration of CAD resources and nongeometric information is just the type of multimedia that WWW browsers support well. Design specifications can be included as HTML text. Early paper sketches or artistic renderings of the final product can be included as images. Animations that depict the manufacture, assembly, or use of the product can be included as movies. Parts catalogs can be hyper-linked, thus allowing easy integration and updating of outside information. Finally, plug-in technology allows external applications to run directly within a page.

Hyper-linked documents also form a natural hierarchy. This implicit hierarchy is well suited to maintaining a design history. Links from documents describing the product goals and requirements might point to documents that contain early sketches of the design. This in turn contains links to designs for the major components of the design. These are, in turn, entire hierarchies that describe the history of the design of each component.

WWW technology appears to be an appropriate medium for integrating all the pieces of DRIVE (and the EDW). Other technologies may also meet the above 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 EDW 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 DRIVE to a wide user base is possible.

Another important step in realizing DRIVE, and the EDW, is making Alpha_1 work with the other components. Once again, one would like to maximize code reuse. Although one option is to reimplement Alpha_1 in some other web enabled programming language such as Java or VRML, this would require a significant redesign of the system. Such redesign can be limited by using browser plug-ins instead. However, Alpha_1 must still be modified to work with the browser plug-in. A plug-in will allow Alpha_1 clients to run within a WWW browser, but this is not enough. It is also necessary to add some knowledge of the WWW to Alpha_1.The next sections explore the required modifications to Alpha_1 as well as some of the other technology that makes DRIVE possible.

4.2 Model Views

One component of an EDW document is a model. That is, the document has a picture of an model, along with the associated Alpha_1 model object graph, embedded in the document along with other design information. Alpha_1 display clients are quite capable of displaying a model, but an embedded model in an EDW document will usually have some additional restrictions placed on it. For example, the designer may want to show only certain parts of the model. 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 an EDW 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 EDW page. One method a designer can employ to capture this information is to run a viewer, set up the viewpoint and display qualities as desired, and capture a screen dump for use in the EDW document. Although feasible, this approach does not yield a "live" model in the document.

The ability to interact with models from within a document is an important aspect of the EDW. Many documents will contain information, such as descriptions of certain pieces or qualities of a design, that dictate a particular presentation of the model. However, such a presentation is only an initial one to provide a context or point of reference for the nongeometric information in the document. The user can also benefit from being able to interact with the model, to change its 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 an EDW 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 4.4. 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, DRIVE implements *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 have been applied to the model. Display properties are properties that a viewer can apply to a model, but 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.

4.2.1 The Alpha_1 Implementation of Model Views

Model views are implemented within the Alpha_1 display library; therefore all viewing clients have support for them. A model view consists of state information that is maintained within a display library window object. Display properties such as surface shading, surface normals, etc., are window-specific and therefore must be stored with each window. A window also needs to know 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 binary input/output mechanism.

4.2.2 The Alpha_1 Display Library

The Alpha_1 display library is closely tied to model views. During the period when model views were under development, the display library was redesigned to eliminate redundant structures and provide more power to viewing clients. This yielded an excellent opportunity to built model views directly into the display library as opposed to placing them on a separate layer above the library.

The new Alpha_1 display library is completely centered around the model graph. Actions such as drawing and picking are supported via traversals of the model graph. Properties inherent to the model (e.g., color and texture) are stored using the model attribute capabilities of Alpha_1 models. Other model display properties that are specific to a window are stored with the window itself, and are accessed via a table that associates the properties with the model objects.

The display library also takes advantage of the powerful binary input/output mechanism of Alpha_1 by making windows objects that can be read or written just like any model. This greatly simplifies the implementation of model views, since a model view is just a window object. In other words, most of the model views implementation is really built in as part of the display library itself. The architecture of the display library is shown in Figure 4.1. Generic window functions and display traversal are handled by the device independent portion of the library. This portion

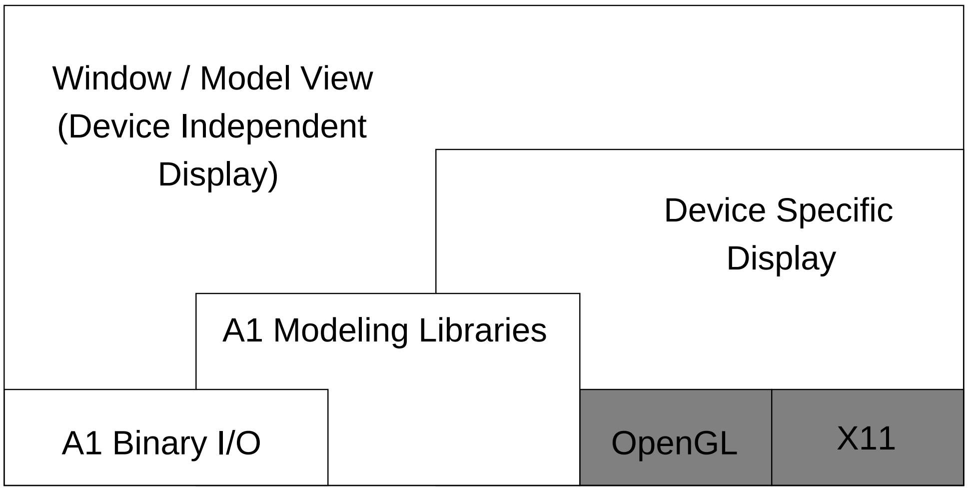

Figure 4.1. The Alpha_1 display library architecture.

of the display library also handles all management of window properties along with any other data that is required for model views. The generic portion is designed in a manner that allows the Alpha_1 binary I/O library to read and write windows objects. Device dependent operations such as the actual execution of drawing code are passed through the generic portion to a device manager object. The device managers implement display for OpenGL and X11.

4.3 Hyper Links

A model view is essentially an "instance" of a model. While this is sufficient to allow the designer to present a model within an EDW 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 capture this hierarchy, individual EDW 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 *embedded* within a WWW document and therefore must provide a mechanism to activate links to other model views or EDW documents. Hyper links are built into the Alpha_1 display library 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, DRIVE will load either a new model view or an entirely new EDW document as specified in the link.

Hyper links are supported through existing Alpha_1 features called model attributes. Model attributes allow nongeometric information to be associated with any Alpha_1 model data structure. The display library handles intersection detection since it is already needed to support basic viewing actions such as selection. Custom user-interface code checks for a link attribute when the user selects a model. If a link attribute is attached to the selected model, then DRIVE carries out the action specified in the link.

4.4 Browser Integration

One of the interesting features of DRIVE 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_1Ḣowever, this is not an attractive approach since it would not allow DRIVE 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 EDW. Since Alpha_1 is such a large system, one of the main goals for integrating it with the web is to minimize the amount of code that must be rewritten.

4.4.1 Java

Java is an Internet ready language that was designed by scientists at Sun Microsystems. Java is quite similar to C++, the language in which most of Alpha_1 is written. Its designers attempted to improve or remove some of the less frequently used and more confusing features of C++. Furthermore, Java is designed to be used over a network. It is compiled to a virtual machine byte code, which is then interpreted appropriately on each platform. The result is a highly portable language that has built-in support by nearly every web browser available today.

Java seems like an appropriate choice for integrating Alpha_1 with the Web, but there are a number of reasons why this has not been chosen in the current approach. First, Java is essentially an interpreted language. The virtual machine byte code does run faster than interpreted C++ would, but Java still runs significantly slower than compiled C++. This limitation is being addressed, as is evidenced by the emergence of "just in time" compilers that compile Java byte code to native machine code on the fly. However, the versions of such compilers that available today have not improved performance enough to make Java a viable language for an application as large as Alpha_1Second, sophisticated graphical user interface toolkits are not yet available for Java. Although Java does provide a toolkit, the Abstract Window Toolkit, it is generally considered to be the weakest part of Java. The AWT simply lacks the power and flexibility that Alpha_1 programs have come to rely on. Finally, Alpha_1 viewers make heavy use of hardware accelerated three dimensional graphics libraries such as OpenGL. Bindings for these libraries are not yet available for Java.

Java is a promising language that has the potential to be as widespread as C and C++ are today. However, its future is still uncertain. Although the web is now proliferated with thousands of Java "applets," the author is unaware of any applications on the scale of Alpha_1 that have been written using Java. For these reasons, the reimplementation of Alpha_1 in Java is a long-term project that should be considered once Java is slightly more mature.

4.4.2 VRML

The Virtual Reality Modeling Language is a standard that was designed in order to bring three-dimensional graphics to the web. The first revision, VRML 1.0, was based on the Open Inventor format, produced by Silicon Graphics, Inc. Open Inventor is a C++ class library that provides generalized modeling capabilities. Open Inventor models are scene graphs made up of Open Inventor nodes. In addition to allowing user-defined nodes, the library predefines a large number of nodes, actions that can be applied to those nodes, and engines that supply data to graph nodes.

VRML 2.0 diverged somewhat from the Open Inventor framework. Most of the changes involved adding nodes, actions, engines, etc. that give more animation and sound capability. In addition, the standard was reengineered to allow for more efficient implementations on PC architectures. The underlying scene graph concept still remains in VRML 2.0, and many of the nodes are still *similar* to their Open Inventor ancestors. For the most part, the changes only add functionality, leaving it possible to write a translator which can convert VRML 1.0 scene graphs to VRML 2.0.

On the surface, VRML appears to be a good candidate for integrating Alpha_1 with web browsers. The scene graph structure of VRML models is quite similar to Alpha_1 models, which are also organized into a scene graph. However, the Alpha_1 model graph is also a dependency graph. Nodes in the graph store construction information such as the function that constructed the node, their prerequisites—the nodes that were needed in order to construct the geometry of a node, and their dependents—the nodes that use this one when constructing their geometry. In fact, the model graph is really a skeleton that defines the structure of a model. The actual geometry is created by "firing" constructor functions that are stored with the model graph nodes. The result is somewhat volatile geometry nodes that can change whenever the prerequisites of their creators change. This information is somewhat difficult to represent in VRML. Open Inventor engines might be suitable for representing the dependency graph structure of Alpha_1 but not without significant restructuring of the Alpha_1 model graph structures. Furthermore, Open Inventor engines are not a part of VRML.

VRML provides the *PROTO* directive for extending the set of VRML nodes, and the *ROUTE* directive for linking fields together. However, this extension mechanism is not as flexible as the one available in Open Inventor. The underlying problem is that VRML is little more than a behavioral specification and file format. Since VRML files are meant to be displayed on machines with many different architectures, there is no way to allow outside code to be plugged in. Any extensions must be entirely contained within a VRML file. The functionality of Alpha_1 simply cannot be implemented using VRML alone.

Clearly VRML will not work for integrating Alpha_1 with the WWW. However, VRML is quite suitable for *exporting* Alpha_1 to the web. That is, Alpha_1 can support VRML as a file format. Although the dependency information would not be included, a VRML version of an Alpha_1 model can still be useful for distributing to a wide audience. The EDW can take advantage of this by serving documents in multiple formats. Users who do not have the Alpha_1 plug-in or a VRML browser could be served EDW documents with static images in place of live models. If the browser supports VRML and the user does not have the Alpha_1 plug-in installed, then VRML versions of the Alpha_1 models would allow the user to see a three-dimensional model that can be manipulated in a limited manner.

4.4.3 ActiveX

ActiveX is closely related to the Object Linking and Embedding technology pioneered by Microsoft. Both are based on the underlying COM technology that allows objects to communicate with applications in which they are embedded. ActiveX provides many of the same features as OLE, but it is optimized for Internet use. ActiveX controls are interactive objects that can be embedded in a web page, thus allowing the integration of user applications with the world wide web.

ActiveX is a potentially good choice for allowing Alpha_1 applications to run within a web browser for many of the same reasons as Netscape plug-ins. Netscape plug-ins are described in Section 4.4.4, next. The main advantage that ActiveX controls have over Netscape plug-ins is that they are not tied to the browser. That is, an ActiveX control can be used directly in any application that supports ActiveX. However, this advantage of ActiveX does not outweigh its shortcomings, the most notable of which is the fact that it is not available for UNIX systems yet. Microsoft has released the ActiveX technology to a standards committee, where it will eventually be made open and ported to UNIX systems. At this time, though, UNIX ports of ActiveX are only in development.

4.4.4 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 that specifies a series of functions that the browser will call when a plug-in is to be initialized, viewed, and unloaded. The plug-in architecture also specifies a new HTML tag, "EMBED." that indicates the location, size, and arguments to a plug-in. When the browser starts, it initializes all shared libraries in its plug-in search path by calling an initialization function that each plug-in must define. Each plug-in uses this initialization function to declare the MIME type(s) it supports in addition to any initialization that needs to be conducted for all instances of the plug-in.

An instance of the plug-in is created when the browser loads a page containing an EMBED tag. The embed tag must contain enough information for the browser to determine the MIME type of the plug-in to be instantiated. The MIME type must be either explicitly declared using the TYPE option of the embed tag, or determined by the server based on the file extension of the URL listed in the SRC option of the embed tag. Once the MIME type is known, the browser calls a function defined by the plug-in associated with the previously determined MIME type to create an instance of itself. Any arguments specified in the EMBED tag are passed along. When the browser is no longer displaying the page with the EMBED tag, the instance of the plug-in is deleted by calling another function defined by the plug-in. Finally, a shutdown function is called when the last instance of a plug-in is deleted. If another instance of the plug-in is created, then the plug-in global initialize function is called once again and the process repeats.

The Netscape Plug-in Architecture has a shortcoming in that plug-ins are created for a specific machine architecture and are therefore not as portable as Java applets. Also, the library or libraries associated with a plug-in must be 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 is that it has to be downloaded only once. This can be a big win for plug-ins that are large, as is the Alpha_1 plug-in. VRML files and Java applets are both stored in the browser's cache. Were Alpha_1 to take advantage of either of these technologies, the data to be downloaded and stored in the cache is likely to be quite large. Since most browsers allocate a fixed size cache, the data downloaded for the document would occupy a significant portion of the cache. The browser would probably either not cache the data at all, or flush it from the cache prematurely as other newer data needed the cache space.

Since the Alpha_1 plug-in is so large, local caching is not an effective option and integration strategies that rely on browser caching would suffer because of this. Thus, local storage of the plug-in is actually a benefit for its regular users. Occasional users can also benefit form local storage of the plug-in. However, these users might not want to download such a large plug-in just for occasional use. Although DRIVE does not support this feature, the EDW could accommodate such users by serving, upon request, VRML models or static images instead of documents referencing the Alpha_1 plug-in.

Although the Netscape Plug-in Architecture is not as portable an integration option as Java or VRML, it is used in DRIVE for the following significant reasons. First, the plug-in is stored locally. The Alpha_1 libraries are very large, so this is a big win in terms of download time. The performance of DRIVE 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). Therefore, the system suffers little performance penalty when running as a plug-in. Finally, the Netscape Plug-in Architecture is the best trade-off of ease of integration versus portability and level of integration. Most of the other integration options require significant modification and rewriting of Alpha_1 source code. This may be acceptable for longer term work on the EDW, but in the prototyping stages for which DRIVE is designed, ease of integration and modification is a must. The Netscape Plug-in Architecture provides this. However, some modifications to Alpha_1 are still required with this approach. More details on these modifications are provided in Section 4.4.5.

4.4.5 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 EDW. 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 EDW in mind. For example, as the EDW 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 [8] toolkit, 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 needs 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 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 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,

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

these toolkits are both design 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 Netscape browser uses 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 EMBED tag is the primary mechanism for initial communication between EDW 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 DRIVE. 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 DRIVE 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 func-

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

tionality through different MIME types. This approach will be useful as the EDW 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 EDW to be available. For example, a "lightweight" plug-in might only 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 EDW. This plug-in would support all MIME types discussed here. The plug-in created for DRIVE supports only the `model/x-a1` type. For development of the EDW, this plug-in also supports Alpha_1 model editing, but not through any special MIME type.

The primary MIME type used by documents in the DRIVE system 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`. Although it is not supported in the initial version of the plug-in, this type will allow the Alpha_1 unique object identifier to be specified as parameters 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. This is essentially the same as specifying an object file, except that the data for the objects is not contained in the file itself. Finally, the MIME type `application/x-a1-edit`, also not yet supported by DRIVE, will indicate that the plug-in should start a `c_shape_edit` server in the plug-in window to serve as a rendezvous for 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 DRIVE. As the EDW is developed, however, it may be desirable to allow the document server to specify the script along with the document itself.

CHAPTER 5

CONCLUSIONS

CAD/CAM systems have allowed designers to significantly reduce the time required by the design process, therefore allowing a number of possible advantages including shorter time to market, timely product releases, reduced design cost, reduced capital cost, superior product design, and smaller lot sizes. Although highly effective, CAD/CAM systems can be improved by adding more support for the design process itself in addition to the current support for individual stages of the process. This research provides a first step in that direction.

DRIVE integrates Alpha_1, a powerful parametric CAD/CAM system, with the World Wide Web through a plug-in version of Alpha_1 modeling and viewing programs. The system, a prototype for the EDW, also brings some web-like behavior to Alpha_1 with the addition of hyper-linked model views. Together, these tools allow the designer to record a design history in an easily navigable document. By utilizing existing WWW technology, the prototype is able to take advantage of a number of useful features of the web. First, web browsers are well suited to including the large amounts of nongeometric design information such as specifications, formulae, sketches, and design rationale descriptions. Second, the web is pervasive, and the familiarity that designers are likely to have with web browsers makes DRIVE an intuitive tool. Finally, by making this information available in a web browser, designers can easily distribute the design history document to a wide range of people. The Alpha_1 plug-in with model views extends web navigation into the Alpha_1 design system, thus allowing the designer to include design history information such as levels of detail directly in the model.

DRIVE is a useful tool in its own right, but it was designed to be part of a larger project. DRIVE is in effect a first generation prototype for the Engineering Design Web. The plug-in and linked model views created for DRIVE are a framework for the EDW. The EDW is a new concept, and it will most likely change considerably as it becomes a reality. The building blocks provided by this research will help advance the EDW to a point where researchers can begin to discover exactly what it should be.

REFERENCES

[1] Banks, M. J. A user interface model and tools for geometric design. Master's thesis, University of Utah, Salt Lake City, UT, 1989.

[2] Boothroyd, G. Product design for manufacture and assembly. *Computer-Aided Design 26*, 7 (1994).

[3] Engineering Geometry Systems. *Alpha_1 User's Manual*. Salt Lake City, UT, 1992.

[4] Gui, J.-K., and Mäntylä, M. Functional understanding of assembly modeling. *Computer-Aided Design 26*, 6 (1994).

[5] Hwang, and Ullman. The design capture system: Capturing back-of-the-envelope sketches. *Journal of Engineering Design 1* (1990), 339–354.

[6] Krause, F.-L., Ciesla, M., Rieger, E., Stephan, M., and Ulbrich, A. 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* (1995), ASME, ASME, pp. 667–685.

[7] Kress, H. Shared 3d viewer, May 1997. <http://www.igd.fhg.de/www/igd-a2/applications/3dviewer_e.html> (Nov. 1997).

[8] Ousterhout, J. K. *Tcl and the Tk Toolkit*. Addison-Wesley, 1994.

[9] Singh, N. *Systems Approach To Computer-Integrated Design and Manufacturing*. John Wiley & Sons, Inc., 1996.

[10] Sturgill, M. M., Cohen, E., and Riesenfeld, R. F. Feature-based 3d sketching for early stage design. In *Proceedings of the Computers in Engineering Conference and the Engineering Database Symposium* (1995), ASME, ASME, pp. 545–552.

[11] Tsai, C.-S., Chang, K.-H., and Wang, J.-Y. Integration infrastructure for a simulation-based design environment. In *Proceedings of the Computers in Engineering Conference and the Engineering Database Symposium* (September 1995), ASME, ASME, pp. 9–20.

[12] Ullman, D. G. *The Mechanical Design Process*. McGraw-Hill, Inc., 1992.

[13] Wang, F.-C., Richards, B., and Wright, P. K. 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* (1995), ASME, ASME, pp. 131–146.

[14] Woyak, S. A., Malone, B., and Myklebust, A. An architecture for creating engineering applications: The dynamic integration system. In *Proceedings of the Computers in Engineering Conference and the Engineering Database Symposium* (1995), ASME, ASME, pp. 1–8.

[15] Zeleznik, R. C., Herndon, K. P., and Hughes, J. F. Sketch: An interface for sketching 3d scenes. In *Computer Graphics* (1996), ACM, ACM Press, pp. 163–170.

[16] Zhang, S., Hou, X., and Wang, Z. Step application protocol interoperability for cad/capp integration. In *Proceedings of the Computers in Engineering Conference and the Engineering Database Symposium* (1995), ASME, ASME, pp. 687–690.