Proceedings of DETC'02
ASME 2002 Design Engineering Technical Conferences
and Computers and Information in Engineering Conference
Montreal, Canada, September 29 - October 2, 2002

DETC2002/CIE-34463

ARCHITECTURE AND IMPLEMENTATION OF A DESIGN REPOSITORY SYSTEM

Simon Szykman
Manufacturing Systems Integration Division
National Institute of Standards and Technology
100 Bureau Dr., Stop 8263
Gaithersburg, MD 20899-8263
szykman@nist.gov

Keywords: design repositories, information modeling, knowledge representation, product development

ABSTRACT

This paper describes the design and development of a design repository software system. This system is a prototype implementation intended to demonstrate the role of design repositories as part of a vision for the next generation of product development software systems. This research involves not only the creation of a prototype software system, but is part of a broader effort that also includes the development of a core product knowledge representation, and that seeks to address terminological and semantic issues associated with computer-aided product development. This paper focuses on the interfaces that have been developed to support authoring and navigation of the product models stored in design repositories, as well as the software architecture and associated rationale that provide the framework on which the system is built.

1 INTRODUCTION

The CAD/CAM/CAE (computer-aided design/manufacturing/engineering) software industry is ultimately a customer-driven one. It is therefore expected that as needs mount, a new generation of tools will emerge to address these needs. As the complexity of products increases and product development becomes more distributed, newly emerging software tools will begin to cover a broader spectrum of product development activities than do the traditional mechanical CAD systems. Accordingly, the ability to effectively and formally capture additional types of information will become a critical issue.

This research involves the development of a vision of next-generation product development systems. This work does not attempt to promote specific tools, but rather seeks to provide an information modeling infrastructure and implementation framework after which new systems can be modeled. The high costs of poor interoperability among today's computer-aided design tools are likely to be significantly compounded in the future if the problem remains unaddressed. This technical thrust addresses a fundamental problem whose solution can impact literally billions of dollars of costs to industry. It is hoped that sufficient diffusion of these concepts into industry will provide a foundation for improved interoperability among software tools in the future.

The NIST Design Repository Project involves research toward providing a technical foundation for the creation of design repositories—repositories of heterogeneous knowledge and data that are designed to support the representation, capture, sharing, and reuse of corporate design knowledge. The roots of the NIST Design Repository Project involved a small effort to develop a software prototype to demonstrate the use of a knowledge representation language created as part of CONGEN (CONcept GENeration) [1], a system that aimed to provide partially automated support for engineering design. An industry workshop held at NIST in November 1996 [2], which identified a number of industry priorities important to future knowledge-based design systems, led to the expansion of the small initial project into a broader research effort.

The infrastructure being developed within the NIST Design Repository Project consists of formal representations for design artifact knowledge, web-based interfaces for creating and browsing design repositories, and facilities to allow searching of repositories using concepts that have engineering design relevance, such as product function. A variety of research activities have been undertaken within the scope of this project since its inception. These include:

- Generation of a model of the flow of design information in product development [3], an effort that examined several

product development processes and proceeded to develop a generic model of the flow of product information independent of any one design process model.

- Development of a Core Product Model [4, 5], a knowledge representation that supports a more comprehensive capture of product information than do traditional CAD systems. The Core Product Model includes concepts such as explicit modeling of engineering function and associated flows (e.g., energy flows), physical and functional decompositions, mappings between physical structures and function, and various other kinds of relationships among these entities.
- Creation of interfaces for authoring, editing, and browsing design repositories that are easy to use and effective in conveying information that is desired.
- Implementation of a web-based software framework to support distributed access to product knowledge stored in design repositories.
- Development of taxonomies of standardized terminology to help provide consistency in, and across, design repositories, as well as to facilitate indexing, search, and retrieval of information from them.

This paper focuses on the current Design Repository System prototype. The general concept behind the Design Repository System remains largely unchanged between the previous and current system implementations. However, the current system embodies fundamental implementational changes that were made based on insights gained from the earlier prototype. The most significant of these changes are (1) a change in the type of client-server architecture used, (2) a change from CGI (Common Gateway Interface) scripts to Servlets to provide functionality on the server-side, (3) a corresponding change from C++ to Java as the main development language platform, (4) a change from an object-oriented database management system to a relational database management system for the back end, and (5) extensive redesigns to the interfaces based on usability testing.

The next section discusses prior work related to this research. Section 3 presents the main Design Repository System interfaces: the Design Repository Browser and the Design Repository Editor. Section 4 describes the system architecture and its implementation. Section 5 provides a discussion and presents areas for future research.

2 RELATED WORK

Traditional CAD systems are limited primarily to representation of geometric data and other types of information relating to geometry such as constraints, parametric information, features, and so on. The engineering design community has been developing new classes of tools to support knowledge-based design, product data management (PDM), and concurrent engineering. When contrasted with traditional CAD tools, these new systems are making progress toward the next generation of engineering design support tools. However, these systems have been focusing mainly on database-related issues and do not place an emphasis on information models for artifact representation (e.g., [6, 7, 8, 9]).

Furthermore, although these systems can represent some kinds of non-geometric knowledge (e.g., information about manufacturing process, bills of materials, catalog/supplier data, etc.), representation of the artifact itself is still generally limited to geometry. This impacts the utility of a range of software tools used in engineering industry. As an example, the lack of a formal product representation that includes function, behavior and structure has been identified as a shortcoming of existing PDM systems [10].

Because of industry's increasing dependence on knowledge, rather than simply geometric data, the product modeling system developed in the NIST Design Repository Project embodies an artifact representation that encompasses a broader engineering design context. This context includes representation not only of geometry, but also of function, behavior, physical and functional decompositions, relationships among these various entities, and so on.

The artifact modeling representation used in this research has its roots in earlier work toward developing an object-oriented representation format that provides a high level division into form, function, and behavior [1, 11]. The fundamental division of design artifact knowledge into these three categories is not unique to those systems. This division has its roots in earlier work in intelligent design system development, and has been adopted by other researchers in various research communities. Examples of such work in artificial intelligence community include the qualitative simulation work in [12], behavioral and functional representation in [13], functional representation in [14] and successive representation from projects such as KRITIK [15] and INTERACTIVE KRITIK [16], the YMIR project [17], and others. Work done in the design and engineering community includes CONGEN [1], the MOSES project [18], the GNOSIS IMS (Intelligent Manufacturing System) project [19], Function-Behavior-State Modeler [20, 21], and a function-behaviour-structure framework [22] and others.

Although many efforts share the form/function/behavior view of product models, this project differs from other efforts in several respects. The representation used in this research extends beyond these three facets of product knowledge, allowing, for example, the representation of requirements and links between requirements and related artifacts or functions. Extensions are being developed to provide links into product development activities beyond conceptual design, such as assembly modeling and process planning. A second distinction between the research presented in this paper and previous work is the development of web-based interfaces to support distributed access to knowledge. Finally, this project has focused not only on the issue of representation of knowledge, but is attempting to address more fundamental issues associated with terminology, and the semantics that accompanies the vocabulary used in product development. These issues, which are often overlooked in the literature, are of critical importance to developing tools to effectively support product modeling. The terminological aspects of this research are beyond

the scope of this paper. They will be described briefly in Section 5, and are discussed at greater length in [23].

A variety of technologies and software architectures have been used to support distributed design and product modeling. Distributed systems may provide interfaces through web browsers or stand-alone applications. The most basic browser-based architectures use HTTP (Hypertext Transfer Protocol) to handle simple communications between a web browser and CGI (Common Gateway Interface) scripts on a web server. This is sometimes referred to as a "stateless" architecture because the scripts are only invoked in response to queries from the client. Since there is no persistence of state on the server, such an approach is too simplistic to support typical product modeling activities.

More sophisticated architectures communicate with applications that run continuously on the server side. For the purpose of distributed product development, these types of architectures often make use of distributed object protocols such as Java/Java RMI (Remote Method Invocation) [24, 25] or CORBA (Common Object Request Broker Architecture) [26, 27, 28]. These protocols may be used for both web browser-based and stand-alone clients.

For some applications, data is managed in a shared database [29], while in others data is passed between applications that have their own local databases. Applications may be able to communicate with one another directly, possibly directly using TCP/IP (Transmission Control Protocol/Internet Protocol) socket connections [30], or alternatively using any of a variety of protocols including (but not limited to) some of the other protocols mentioned above. In other cases applications that are not designed to communicate or interoperate may use wrappers or agents [31, 32] to handle communications. Furthermore, these various ingredients should not be considered to be mutually exclusive, as they can often be combined in different ways. For example, a CORBA-based implementation may or may not make use of Java as a programming language, while a Java-based implementation may choose to use CORBA instead of Java/RMI as a distributed object protocol. Similarly, an agent-based system can be built upon any of a number of infrastructures, including one based on Java/RMI [25] or CORBA.

Still another option for developing product development systems is to build a system using an existing commercial infrastructure. One example of such a tool built using Windchill,[1] Parametric Technologies Corporation's collaboration technology architecture, is presented in [33]. Several other companies are also providing commercial infrastructures for collaborative commerce, including EDS (which now incorporates Unigraphics Solutions and SDRC), MatrixOne, ENOVIA Corporation (a subsidiary of Dassault Systemes that has formed an alliance with IBM), CoCreate, and others.

This discussion is not meant to provide a complete list of all potential technological decisions or solutions involved with designing a product modeling system. The literature review is representative rather than comprehensive. The intent is to convey the fact that many alternatives are available for developing systems. No one approach will be best in all situations. Thus, a sufficient depth of technical understanding of the various alternatives is necessary in order to have system requirements, rather than arbitrary decisions, drive the system development process.

3 DESIGN REPOSITORY SYSTEM INTERFACES

3.1 Overview of Interface Functionality

The various web-based user interfaces serve as links between the user remainder of the Design Repository System. These interfaces accept requests from the user, and deliver the results of the request via dynamically generated web pages. Web pages include content in HTML (Hypertext Markup Language), and in most cases, JavaScript as well. The use of JavaScript allows some processing of page content to be done on the client side. For instance, the dynamically expandable and collapsible hierarchical product decomposition (illustrated in Section 4) is accomplished using JavaScript.

The Design Repository Browser provides an interface that allows a user to navigate through the body of product knowledge contained in a design repository. The user can navigate through this information along a variety of paths. The user can move up and down the physical hierarchical product decomposition to reach a design repository object[2] of interest. The user can also move from one design repository object to another along links between the objects. For example, when viewing a given artifact (an object that represents a system, an assembly, a component, etc.), the user can move "down" in the hierarchy to one of its constituent parts, or "up" in the hierarchy to a higher-level artifact which has the current artifact as one of its parts. The user can also move along links that do not correspond to the physical hierarchy. For instance, the user can move from an artifact object to an object representing its function, or to other objects representing energy flows of which the current artifact is a source or destination.

The Design Repository Editor allows the user to create product models by authoring new design repository information, as well as modifying existing information in a repository. When viewing a product model from the browser interface, a user that has the appropriate privileges can click on an edit button to enter the editing mode. In edit mode, the user can modify information on the current page. While in this mode, the user also has the ability to create new data entities (i.e., new artifacts, functions, flows, etc.), or to add, change, or re-

[1] Use of any commercial product or company names in this paper is intended to provide readers with information regarding the implementation of the research described, and does not imply recommendation or endorsement by the author or the National Institute of Standards and Technology.

[2] Although the Design Repository System is implemented using Oracle 7, a relational database, the core product model used for representing product knowledge is conceptually an object-oriented model. Thus, although information is physically stored in tables in a relational database, the term "design repository object" (or simply "object") will be used to refer to the data entities that are used to represent a product.

move links between entities. Clicking on a button to return to browsing mode allows the user to confirm changes and commit them to the database. The next section provides a more detailed description of the specifications and implementation for the Design Repository Browser and Editor interfaces.

3.2 Interface Design and Implementation

The Design Repository Browser and Editor interfaces have a frame-based structure where each of several different frames is used to display certain kinds of information. A schematic of this frame-based structure is shown in Figure 1.

The overall layout for the frame set shown in Figure 1 is contained in a static HTML file that specifies the geometry for the various frames. This file also specifies which additional files supply the content for each of the frames in the frame set. Below are descriptions for each of the frames that appear in Figure 1.

- Top Frame: This static frame displays the Design Repository Project banner.
  User Frame: Displays a welcome message including the name of a user when a user is logged in. The User Frame also contains a link to the search tool and the login/logout page.
- History Frame: Displays the list of the objects viewed by the user in the current session and allows quick access to these objects.
- Tree Menu Frame: This frame displays the hierarchical decomposition tree for artifacts (by default) or functions (if selected) for products in the design repository database. The dynamically expandable and contractable hierarchical product structure tree makes use of a third-party script called FolderTree, which is written in JavaScript, and makes use of Dynamic HTML (DHTML) capabilities supported in recent versions of most web browsers. FolderTree is freely available at <http://www.geocities.com/Paris/LeftBank/2178/foldertree.html>.
- Tree Control Frame: Displays controls for the Tree Menu Frame, such as switching between an artifact tree and a function tree or rebuilding these trees.
- Main Frame: The main frame is empty in its initial state. Different JavaServer Pages are used to display content about various objects in a design repository as the user interacts with the system. Editing of objects and authoring of new objects is also done in the Main Frame.
- Control Frame: Contains controls that affect the content of the Main Frame, such as switching between browser and editor mode, or selection of new objects to create in edit mode.

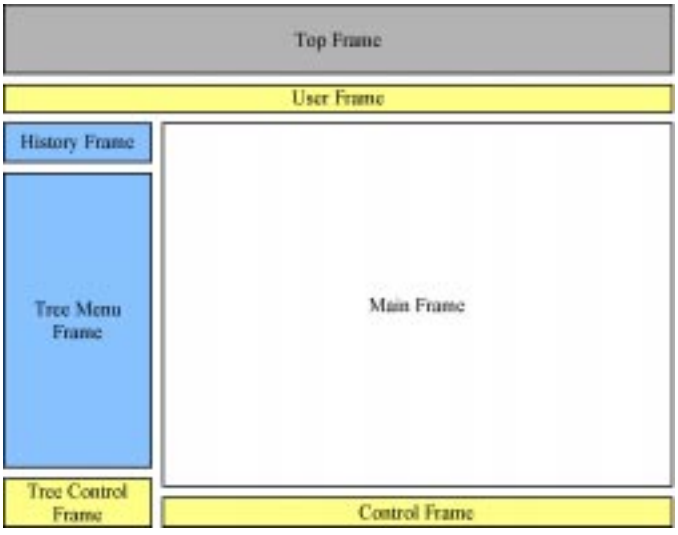

Figure 1. Frame-based structure of the Design Repository Browser and Editor interfaces

Figure 2 shows the welcome page of the Design Repository System interface. The figure shows the various frames and their default content when a user makes an initial access to the system. At this stage, the user has not yet logged in. The user can click on the login link in the User Frame to access a login screen and log in with a user name and password.

Without logging in, a user can view the hierarchical product decomposition tree in the Tree Menu Frame, or can perform a search using the Search Tool. However, a user who is not logged in does not have read privileges to view any additional information. Any attempt to view details of the objects that appear in the Tree Menu Frame, or objects resulting from searches, will automatically bring the user to a login screen requiring a login before information may be displayed. The "edit" button in the control frame is used to switch to editor mode, but as with viewing of object details, a user who is not logged in will be presented with a login screen before being able to use the editor.

Figure 3 shows a screenshot of the Design Repository Browser interface when the user is viewing an artifact. The hierarchical product decomposition tree (shown partially expanded in the Tree Menu Frame at the left) allows a user to expand and view in detail one or more portions of a physical hierarchy, while leaving the rest of the tree collapsed to hide product structure detail that is not currently of interest to the user.

When a design repository object is viewed, all of the information that is related to that particular object is retrieved from the database and sent to the web browser on the client side. Since users typically view an object with an interest in only a subset of this information, the complete object description is generally more than a user wishes to see at once. Nevertheless, retrieving all of the information at once provides a significant advantage in terms of responsiveness of the interface. Although the time required to retrieve the complete object description is slightly longer than the time it would take to retrieve only the subset of information that a user is interested in, this difference is relatively small. Once the information is on the client side, the user can selectively view different subsets of information without the server having to process any new requests from the client. This approach thereby eliminates even brief delays that would be needed for additional

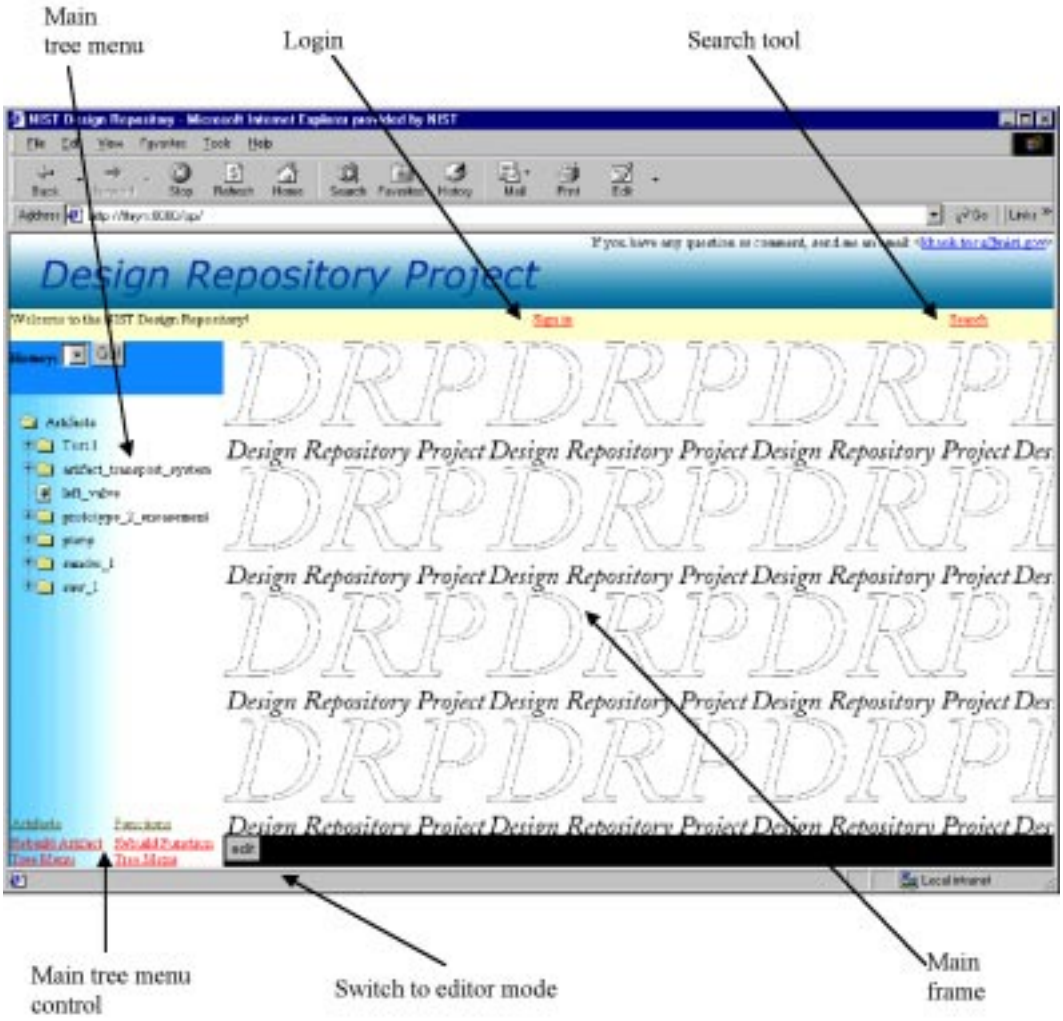

Figure 2. Screenshot of the welcome page

client requests to be processed, information retrieved from the database, and sent back to the client.

To avoid overwhelming a user with an excess of detail, the Design Repository Browser initially displays only a subset of the information that is related to a given design repository object. The detailed information has been sent from the repository to the web browser on the client side, but the information is hidden using an expandable/collapsible display structure implemented using Dynamic HTML. In the interface, black triangles serve as visual cues to show where information can be expanded (when the triangle is pointing to the right) to show more detail, or collapsed (when the triangle is pointing down) to hide detail. In Figure 3, the first two triangles indicate that additional information about the current object and its functions is available. The third triangle is pointing down, indicating that the user has already expanded the view to show additional information about the form of this artifact.

Clicking the "edit" button in the Control Frame switches the interface from Browser mode to Editor mode. A screenshot of the interface in Editor mode is shown in Figure 4. This mode displays much of the same information that the user can view in Browser mode, but changes the interface to a form, allowing the user to edit information that previously could only be viewed.

In addition to editing textual information, in Editor mode the user can also edit links between design repository objects. For instance, the user can assign an existing function object to an artifact, or identify a set of existing artifact objects as being sub-artifacts of the current artifact. This is done by selecting the type of design repository object to be linked to the current object. In editor mode, the user may also choose to create a new design repository object by selecting an object type from the pull-down list in the Control Frame and clicking the "Go" button. Figure 5 shows a screenshot of the Design Repository Editor during the process of creating a new artifact object. Once the user is done editing, clicking the "browse" button allows the user to confirm changes, commits them to the database, and returns to Browser mode.

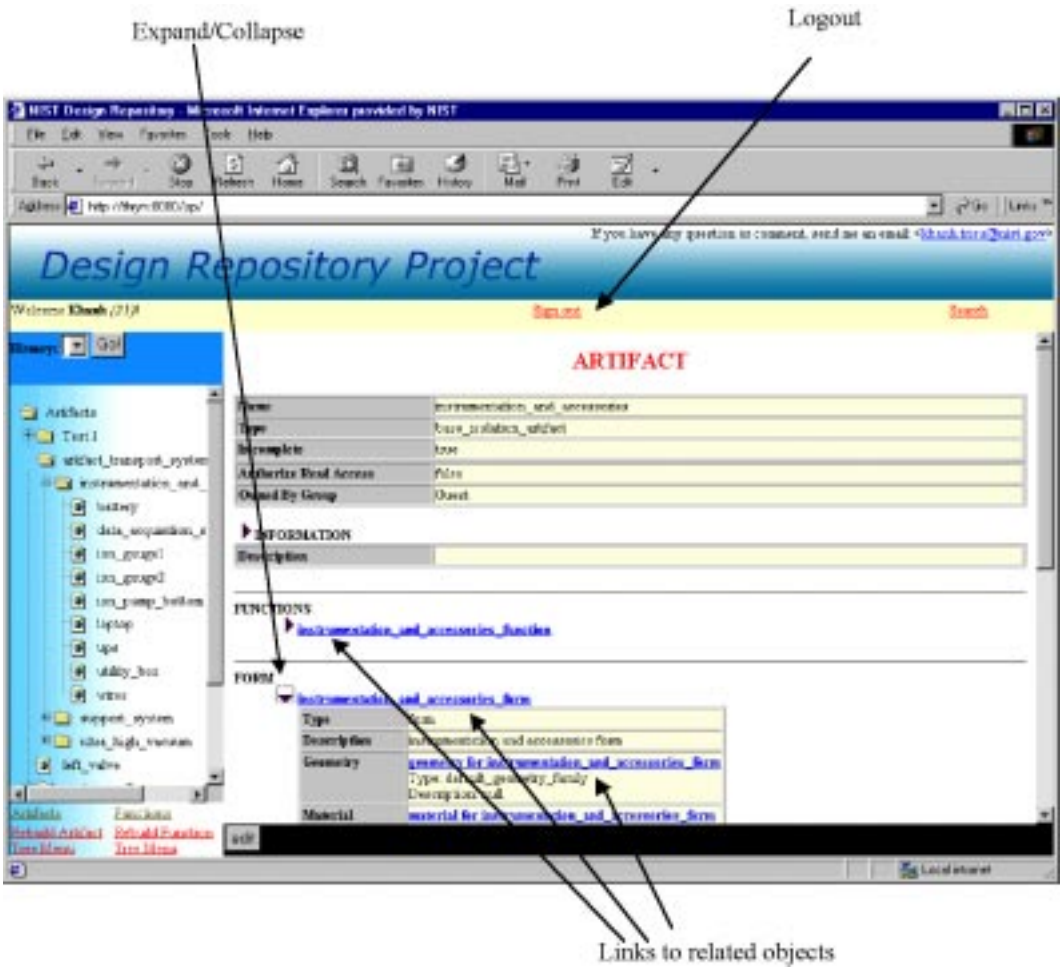

Figure 3. Screenshot of the interface in browser mode

4 SYSTEM ARCHITECTURE DESIGN

On the server side, the Design Repository System makes use of Sun Microsystems' Java Web Server 2.0, Oracle 7 relational database management system, Java 1.2 with Java Servlet API (Application Programming Interface) and JDBC[3] (Java Database Connectivity) API, a web browser (Netscape Navigator/Communicator 4.0 or later, or Internet Explorer 4.0 or later) with JavaScript enabled. On the client side, only the web browser with JavaScript enabled is necessary.

[3] Although the current implementation uses Oracle 7 as the database back end, because interactions with the database are accomplished using JDBC, a standardized Java-based SQL API, it should be possible to substitute any SQL-compliant relational database management system as the back end. It should be noted that not all relational database management systems support the triggers and sequences used for database consistency maintenance in the current Design Repository System implementation. Thus, while a substitution of the database back end should not impact the ability to store and retrieve information from the database, in some cases a change of back end may require that certain consistency constraints either be sacrificed, or implemented in software instead of maintained through the database itself.

4.1 Overview of Functional Components

The main functional components of the Design Repository System architecture are the Request Handler, the Database Exchange Manager, the Database Connection Manager, the Database Management System, the User Interfaces, the User Management System, and the Web Server. An overview of these components follows.

Request Handler: The Request Handler handles the various requests from the user. This component is the starting point for processing any kind of request from the user.

Database Exchange Manager: The Database Exchange Manager is the interface between the web server and the database server, and handles the exchange of data between the two servers. This component uses an object model to store data in memory when retrieving information from the database. The Database Exchange Manager is used by the Request Handler to retrieve data to satisfy user requests.

Database Connection Manager: The process of spawning a new connection to the database is time consuming enough to cause perceptible delays in the responses to user requests. The

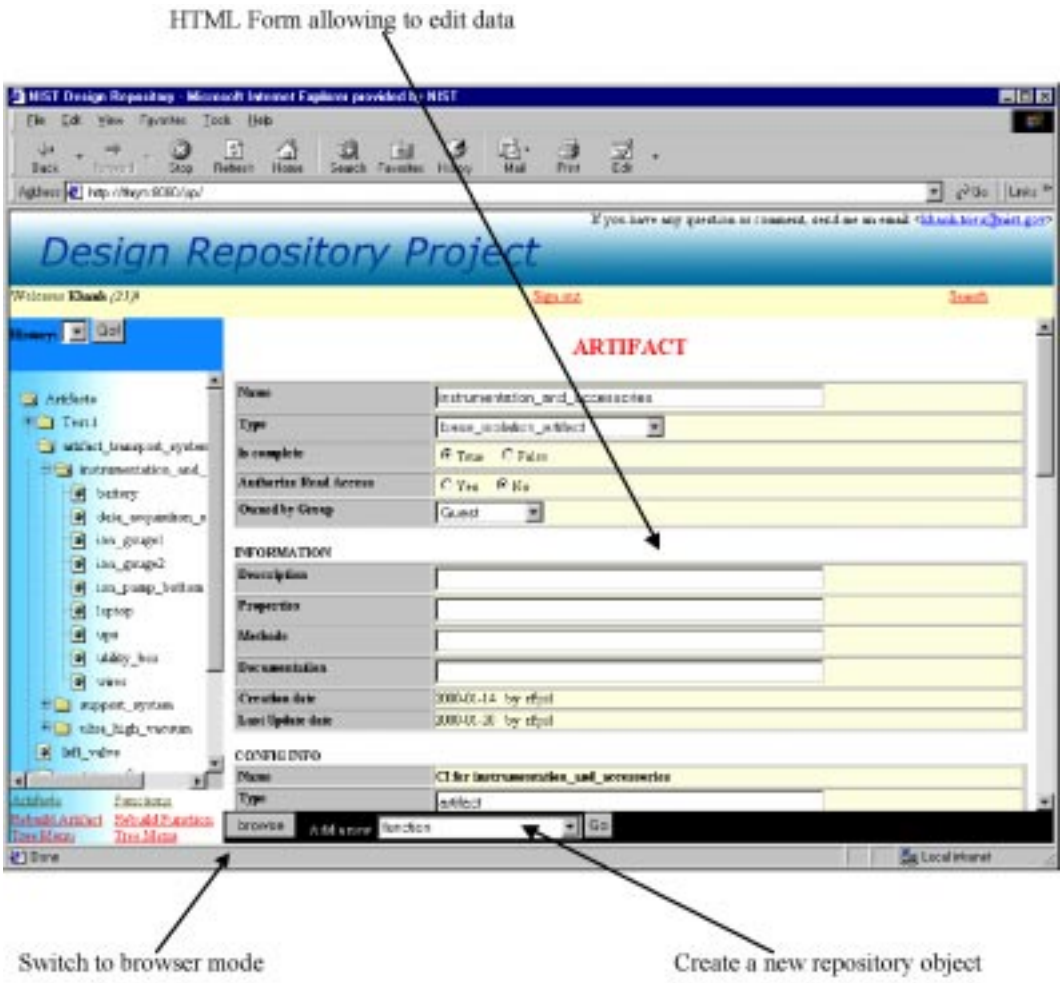

Figure 4. Screenshot of the interface in editor mode

Database Connection Manager is used to dynamically maintain a pool of open connections to the database, so that user requests can be handled without having to open new connections in response to these requests. When a request gets passed to the Database Exchange Manager from the Request Handler, the Database Exchange Manager requests an (already open) connection from the Database Connection Manager.

Database Management System: The database used for the Design Repository System is Oracle 7, a relational database management system (DBMS). Because the underlying representation used for modeling product knowledge is conceptually an object-oriented representation (see [4, 5]), one might argue that an object-oriented database would be better suited to this application. In general, the choice of a DBMS may be driven by numerous factors in addition to the obvious issue of the nature of the information being modeled. In the case of this project, several other important drivers led to the choice of a relational database system for the back-end database.

One factor is the fact that standardized Java-based APIs exist for creating generic (non-vendor-specific) interfaces to SQL (Structured Query Language) compliant relational databases, whereas no such APIs yet exist for object-oriented databases. Because databases from many different vendors are used by companies in industry, it was considered important for this project to not be tied to any one particular vendor's database management system, to show relevance to as large a cross section of industry as possible. The heterogeneity of database systems outside of NIST also could potentially serve as a barrier to collaboration with other groups within the scope of this project. The ability to construct generic interfaces that could communicate with databases from different vendors would make external collaborations easier.

Lastly, if the Design Repository System were developed using an object-oriented database, there might be a perception that this type of DBMS is a requirement for implementing such a system in industry. Because relational databases are much more prevalent in industry than in object-oriented databases, such a perception might inhibit diffusion of this technology into industry. The choice of a relational database system for the back end demonstrates that implementations do not require object-oriented databases, and are possible with relational databases as well.

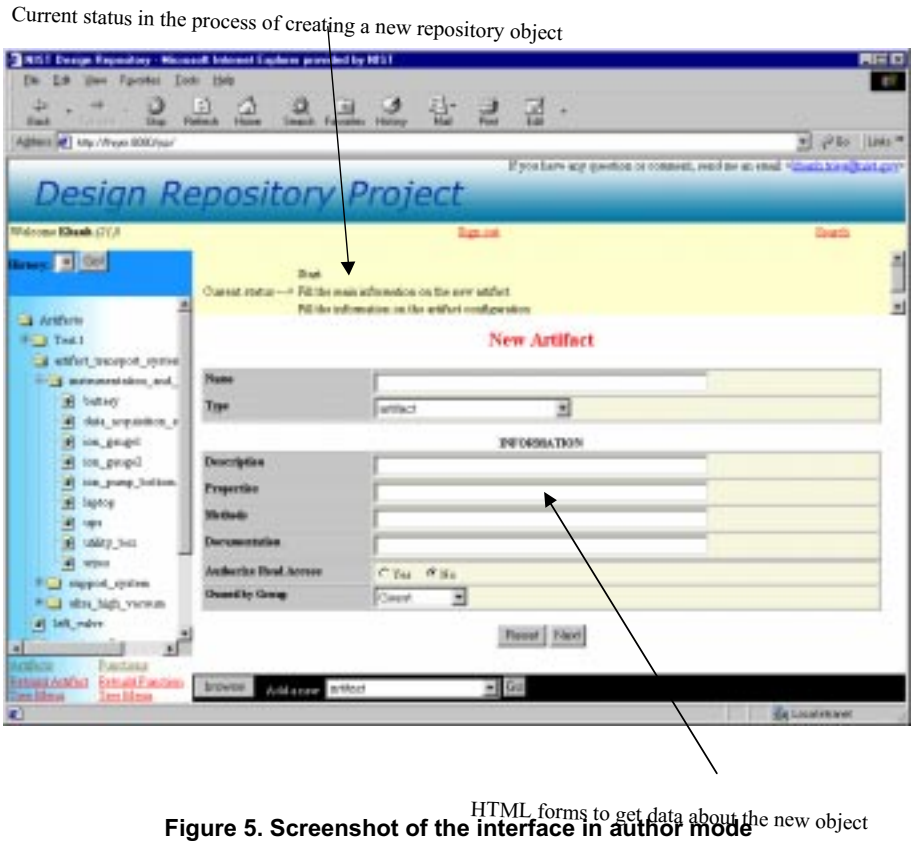

Figure 5. Screenshot of the interface in author mode

User Interfaces: The designer makes use of various user interfaces to interact with the Design Repository System. The Design Repository interfaces serve as the end user's access point to product information that is stored in design repositories, enabling the end user to author, edit, retrieve and view this information. Information supplied by the user is processed and stored in the database. Information retrieved by the user is formatted using HTML and JavaScript, to provide a client-side presentation of information that is readily interpretable by a human user.

User Management System: The User Management System allows administrators to manage access to product information by allowing the creation of groups, the addition and removal of users from groups, and the granting of permissions associated with information access. Based on the groups a user belongs to and the permissions a user has been granted, a user may be denied access to information, a user may be permitted to only view information (read-only privileges), or a user can also be allowed to edit information (read/write privileges). Information for each product is stored in a design repository. Each repository belongs to a group, so that each user can have different permissions (no access, read-only, read/write) for each repository based on the permissions that have been granted to that user by the administrators of each group the user belongs to. Because the implementation of user management systems is relatively routine and does not press the state of the art, this aspect of the Design Repository System will not be discussed further in this paper.

Web Server: The Web Server is used to handle communications between the user at the client side and the main part of the Design Repository System on the server side using HTTP. The web server used in the current implementation is Sun Microsystems' Java Web Server 2.0.

The interaction between the various functional components is shown in Figure 6. Regardless of the activities users involved with (browsing/editing information, searching repositories with the search tool, or performing user management activities), users interface with the Design Repository System using a web browser as a client. Thus, although these activities are functionally different and are implemented as different components, the interfaces that users interact with to accomplish these activities are not individually shown in Figure 6. The interface output logic box as it is shown in Figure 6 is not, strictly speaking, a single separate component, but rather is used to illustrate the fact that information is formatted (using HTML and JavaScript) for web-based viewing before being sent to the user.

Section 4.2 provides additional information regarding the design of the client/server architecture. Sections 4.3 - 4.5 describe the Request Handler, the Database Exchange Manager and the Database Connection Manager in greater detail. Because the DBMS and the web server used for the NIST Design

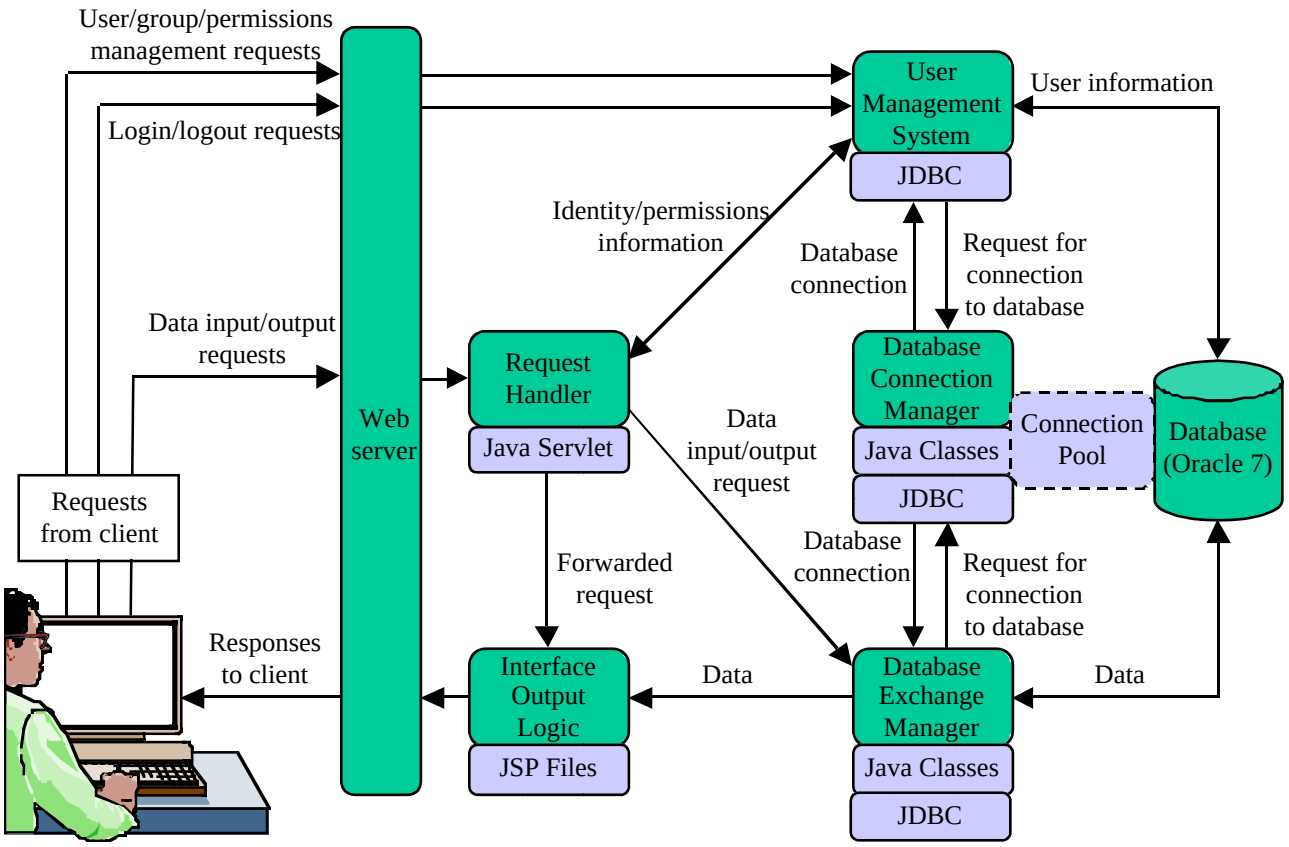

Figure 6. Interaction between functional components

Repository Project are commercial systems and weren't developed as part of the project, this report does not include technical or implementational descriptions of these systems. Technical information about Oracle 7 can be found at `http://docs.oracle.com/`. Technical details regarding Sun Microsystems' Java Web Server 2.0 can be found at `http://docs.sun.com/`.

4.2 Client/Server Architecture Design

The Design Repository System is based on a three-tier client/server architecture. Three-tier architectures emerged to overcome the limitations of two-tier architectures, which include poor performance with large numbers of users, and limited flexibility. In a two-tier architecture, one tier is the client and the other tier is the server-side DBMS. In a three-tier architecture, the third tier is an application that sits between the client and the database server.

There are a variety of ways of implementing this middle tier, such as transaction processing monitors, message servers, or web/ application servers. The middle-tier in a three-tier architecture can be used to perform such functions as queuing, application execution, database staging, and others. Three-tier client/server architectures have been shown to improve performance for groups with a large number of users (even as high as thousands) and provide greater flexibility than a two-tier approach. Today, most sophisticated web-based applications that involve data exchange are based on three-tier client/server architecture.

Figure 7 is a simplified view of Figure 6 that illustrates the three tiers in the Design Repository System architecture. In the Design Repository System, the client is the web browser that runs on the user's computer. The third tier is the Oracle 7 database management system. All of the other components described in Section 4.1 form the middle tier.

In practice, various components may reside on different machines. The web server may be on one machine, the rest of the middle tier application on another, the database server on yet another, and depending on the size of the databases, the data itself may be distributed among additional machines. Standard protocols such as TCP/IP and HTTP exist for handling communications among distributed components, so dealing with distributed components does not greatly add to the difficulty of implementation if the application development is done using techniques that conform to these standards.

The desire for a three-tier architecture, the need to conform to standard protocols, and the intended usage of the application motivate a number of architectural choices, including which type of client-server architecture to use, which development language to use, which technologies to incorporate into the system implementation, etc.

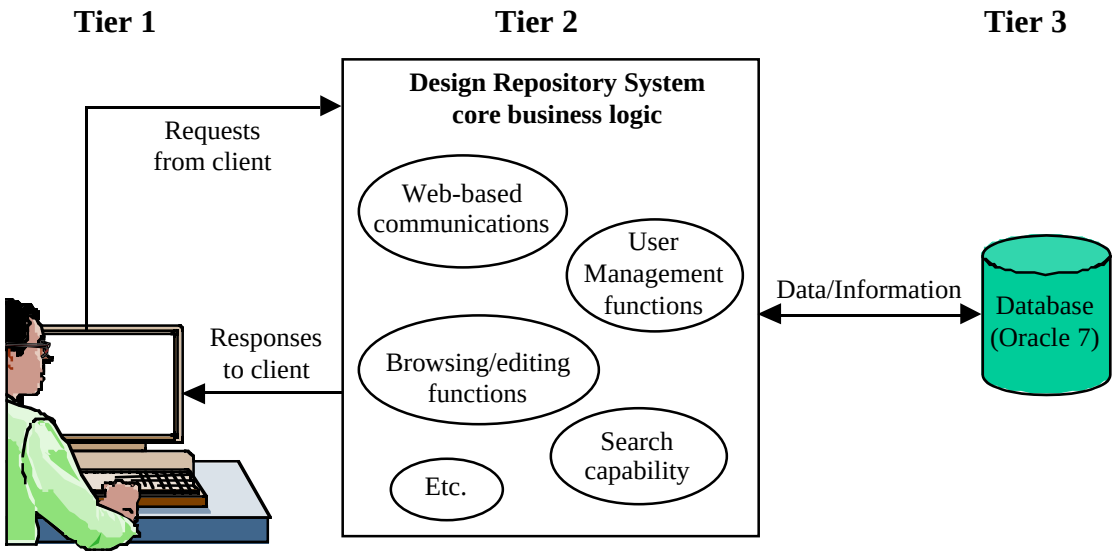

Figure 7. Three-tier architecture for the Design Repository System

Three classes of client/server models are a fat client, a thin client, and an ultra-thin client. These models differ in how much of the "business logic" of the application resides on the client side vs. the server side. Fat clients, which have a significant portion of the business logic on the client side, were more common prior to today's prevalence of web-based applications. A client can easily be "fat" when the client application with the built-in business logic is installed on a local machine. For web-based applications, web browsers provide a standardized client and any business logic must be delivered to that client separately. Fat clients are not commonly used for web-based applications due to delivery issues, such as bandwidth limitations that can lead to impractical download times, and the frequency with which updates are made to business logic in a fast-moving software development world.

The difference between a thin and an ultra-thin client is again the degree to which a portion of the business logic is handled on the client side. Examples of thin clients might be applications which require the downloading of plug-ins or Java Applets, allowing some of the "work," be it computing, data processing, etc. to be done on the client machine. Ultra-thin clients attempt to minimize the processing done on the client machine. The dividing line between thin clients and ultra-thin clients changes as technology evolves. For example, the use of JavaScript (a scripting language supported in most web browsers) in web pages allows web browsers to easily and quickly perform certain types of functions that in the past would have required an Applet to be downloaded in order to accomplish on the client side.

The Design Repository System is implemented using an ultra-thin client/server model. One of the main motivations for selecting an ultra-thin client over a thin client was the recognition that many of the intended users of the type of technology developed in this project would be small and medium sized businesses that would not necessarily have the latest (i.e. powerful) computer hardware available to them. The ultra-thin client minimizes the burden on the client machine.

The second motivation was a desire to support a broad base of potential users in a heterogeneous software environment. For example, one approach to developing a thin (rather than an ultra-thin) client would have been to create a Java Applet-based interface that would be downloaded to the client machine, allowing some of the data processing to be handled on the client machine. However, the Java language is an evolving one. Web browser support for Java 2, the latest version, may be constrained by one's choice of web browser, the version of the web browser being used, as well as the operating system on the client machine. An Applet-based interface would therefore not be accessible by as broad a user base as one which uses only HTML and JavaScript, which is more uniformly supported by web browsers than Java 2.

Among development languages, C++ and Java are the most widely used languages. Although earlier versions of the Design Repository System made extensive use of C++, a decision was made to shift to Java for the latest implementation. This decision was made based on the availability of existing Java-based technologies to support web-based application development, including Java Servlets, JavaServer Page, and JavaBeans.

A Java Servlet is a Java program that runs on the server side, as contrasted with the more familiar Java Applets, which are Java programs that run on the client side. The advantage Servlets have over Applets is that they can enable complex programming, but being on the server side they avoid the download times that would be needed to download large Applets to a web browser. An alternate traditional method for supporting server-side processing is the use of CGI scripts, which spawn processes of short duration for each data access. These processes are terminated once a script execution has completed, and once a process terminates no portion of the

computation remains in active memory. In contrast, Servlets are applications that run continuously, allowing information to be cached in active memory, significantly reducing the time expended by making repeated accesses of the database.

Servlets, being implemented in Java, provide a component-based, platform-independent method for development of web-based applications. Servlets are also more portable than some of the proprietary web server extensions that can be used for application development (e.g., Netscape Server API), as they are not limited to use with a single vendor's web server.

JavaServer Page (JSP) is an open specification developed by an industry-wide effort led by Sun Microsystems, which provides a simplified method for rapidly creating web pages that display dynamically-generated content. The JSP specification defines interactions between the web server and JSP pages, and allows the format and syntax of pages to be defined. JSP pages are compiled into Servlets in order to be used. It is possible to accomplish with Servlets alone that which is done with JSP. The advantage to using JSP is that the JSP separates the form of web pages—the templates that define how they appear—from their content. In applications such as the Design Repository System, where web page templates are static but content is dynamically generated (from a database in our case), using JSP provides a much-simplified development approach to generating web pages.

JavaBeans is a platform-independent component model written in Java. The component model allows developers to author reusable platform-independent software components that can be used for building larger applications. Being a complete component model, JavaBeans provides features such as properties, events, methods, and persistence. Although the concept of reusable software components is not new to JavaBeans, JavaBeans has been developed to enable automated analysis of components, to simplify customization of components, and to provide a foundation for component-based software development using Java.

Both Java Servlets and JavaServer Pages allow server-side programming using the Java language. Based on the roles that these technologies play in software development, three alternatives were considered for the overall system architecture approach: (1) using only Java Servlets, (2) using JavaServer Pages in conjunction with JavaBeans, or (3) using Java Servlets, JavaServer Pages and JavaBeans.

With the first alternative, the middle tier would consist of the web server and Java Servlets, which would handle all of the processing in the middle tier. This processing would include processing HTTP requests coming from the client via the web server, exchanging data with the database server, dynamic creation of web pages containing HTML and JavaScript, and sending them to the client via the web server using HTTP. The advantage of this approach is that it is easy to implement relative to the other alternatives. However, the disadvantage is that because the web page generation code is built into the code for the Java Servlets, subsequent maintenance (e.g. updating of web page templates) can become a burden.

For the second alternative, the middle tier would consist of the web server and a set of JavaServer Page-JavaBean pairs. Each type of object stored in a product repository would be handled by a separate JavaServer Page. HTTP requests would go to the appropriate JavaServer Page via the web server, depending on which type of information is being processed. For instance, a request for product function information would go to one JavaServer Page while a request for a product form would go to another. Each JavaServer Page would form the basis for a web page, and would use an associated JavaBean to retrieve data from the database, providing the dynamic content for the web page containing the requested information.

Since JavaServer Pages are compiled into Java Servlets in order to be used, this architecture is physically nearly identical to the previous one. The use of JavaServer Pages simplifies the task of writing the HTML generation code, making implementation easier than with the first alternative. This approach is not without drawbacks, however. Because the JavaServer Pages include Java code, this approach is not well suited to systems that have sophisticated business logic (complex functionality). Too much Java in the JavaServer Pages can make it very difficult to debug software during development, and longer-term maintenance can still be problematic.

The third alternative is to use Java Servlets, JavaServer Pages, and JavaBeans. With this architecture, Java Servlets are used to handle incoming requests, processing, and other business logic. For a given request, an appropriate JavaBean is instantiated to handle data exchange with the database. Information about the request is passed on to the appropriate JavaServer Page, which then receives the query results from the JavaBean and sends them out to the client in the appropriate format.

This approach provides the intended benefits of Java Servlets, as well as of the JavaServer Pages and JavaBeans combination. More specifically, this architecture effectively separates the business logic (the core functionality of the application) from the graphical user interface definition (the code that relates to formats or templates for viewing and displaying information). Because of this decoupling, these two layers can be developed almost entirely independently from one another. More importantly, once deployed, they can be maintained and updated separately. The business logic can be extended without having to modify the code associated with displaying information, and similarly, the interface can be revised without digging into the core application code.

The only disadvantage of this architecture in contrast to the previous ones is that the implementation itself is more complex as a result of incorporating all of the technologies used individually in the previous alternatives. Because of this increased complexity, a more thorough understanding of the various technologies is necessary in order to achieve a successful implementation. Nevertheless, because of the separation of layers, such an architecture can be easier (though not necessarily faster) to implement. More importantly, the burdens associated with extending the system after initial de-

ployment, and longer-term system maintenance, are considerably reduced.

The choice of architecture was made with the intent of producing an application that would be fast, robust, and easy to extend and maintain. This decision was made with some insight into what would be required to achieve the system functionality that was desired. A system with very modest interface requirements might best be implemented using the first architecture. A system with comparatively simple business logic might be best implemented with the second alternative. Such a system would be easier to implement, and could still be easy to maintain if significantly less core code were needed to provide the desired application functionality.

Based on the requirements of the Design Repository System, the third alternative was selected for this project. Among the three architectures considered, all could provide a fast and robust application but the last alternative would be significantly easier to extend and maintain. The interactions among the various technologies involved is shown in Figure 8. This generic architecture is essentially a higher-level view of the more detailed, component-level view of the Design Repository System architecture that was shown in Figure 6.

4.3 Request Handler

The Request Handler is the entry point to the middle tier of the three-tier architecture. The Request Handler consists of a set of Java Servlets that receive HTTP requests from the client through the web server, and execute Java code to process the requests. The Request Handler first verifies that an incoming request is a valid one. Assuming it is, it then dispatches commands to the Database Exchange Manager and passes information regarding the request to the code associated with the user interface definition so that the results of the query can be constructed, appropriately formatted, and sent back to the user. Responses to requests and/or confirmation of transactions are returned to the user as web pages (formatted in HTML and JavaScript) sent through the web server.

4.4 Database Exchange Manager

The Database Exchange Manager is the interface between the middle tier and the third tier of the system architecture (see Figure 6). The Database Exchange Manager sends SQL requests to the database in order to exchange data, using JDBC a standardized Java-based SQL API mentioned in Section 4. This data exchange is done via an open connection to the database, which the Database Exchange Manager obtains from the Database Connection Manager (discussed below). When data are retrieved from the database, these data are kept in memory using a Java object model that mirrors the Core Product Model (discussed in Section 1) that is used in the NIST Design Repository Project. Once stored in the object model, these data can be accessed by the other components of the Design Repository System. The Database Exchange Manager is implemented using JavaBeans.

4.5 Database Connection Manager

The Design Repository System provides interfaces for creating, retrieving and editing product information stored in a database. These activities involve the exchange of information to or from a database. Many of the actions available to a user through the interface result in communication with the database. Establishing a connection to a database is a time consuming function because the database must allocate resources (such as memory), authenticate the user, set up the corresponding security context, etc.

Calling the establishment of a connection "time consuming" is, of course, a relative characterization. With the network infrastructure that the Design Repository System runs on,

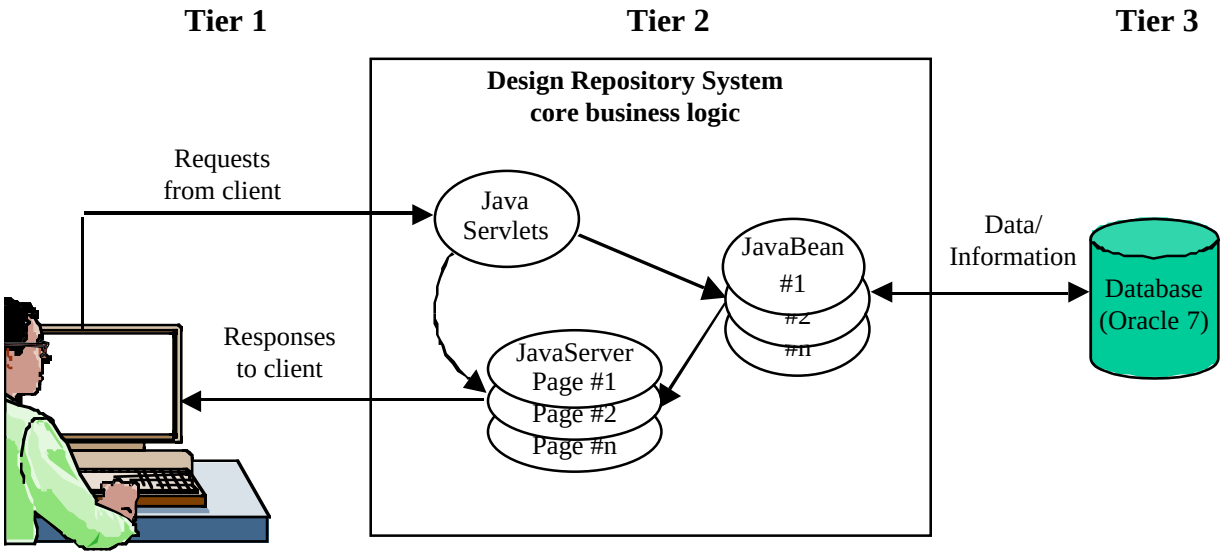

Figure 8. Three-tier architecture with Java Servlets, JavaServer Pages and JavaBeans

and with the web server and database servers being run on separate machines, establishing a connection to the database typically takes one second or longer. As sources on interface design commonly cite 200 milliseconds as being the threshold below which response appears to be instantaneous and above which humans perceive a delay, it is evident that it does not take much of a delay to create unappealing response lags in a human-computer interface. Indeed, the previous Design Repository System implementation did suffer from excessive delays in response to user input, and the time for opening connections to the database server for each query was experimentally found to be one of the most significant contributors to the time lag. The Database Connection Manager was developed to address this issue.

The Database Connection Manager is designed to implement a pooling technique that allows multiple database connections to be established and maintained in a connection pool, so that they can be shared transparently among multiple incoming requests. The Database Connection Manager creates the pool of database connections when the main application of the Design Repository System is initially started up. Consuming the time required to establish connections at startup significantly reduces the overhead on database queries made later in response to user requests. Once a user request is satisfied, the connection is released back into the pool without closing it.

The connection pool starts up with a default number of connections, but is implemented to optimize resource usage by altering the pool size based on demand. The number of connections in the connection pool can be dynamically increased as the number of free connections in the pool drops below some limit, or reduced when the number of unused connections exceeds a specified number. When it is necessary to add connections, establishing new connections takes as much time as it would without a connection manager in place. However, this time usage does not delay the system response to user queries because new connections are added while some unused connections are still available in the connection pool. Thus the connection management is done invisibly to the user without causing unwanted response delays.

5 AREAS FOR FUTURE RESEARCH

This paper has described the functionality, interfaces, architectural design, and implementation of a Design Repository System. The work being done as part of this project is intended to provide a foundation for the development of a new generation of computer-aided product development tools. Although this paper focused mainly on technical aspects of the Design Repository System, the project itself is concerned with contributions at several other levels. These include the development of more comprehensive representations of product knowledge than are available in traditional mechanical CAD systems, the design of interfaces that allow users to navigate complex product models and that deliver information in an easily interpreted format, and efforts to address terminological issues in product development support tools.

The Design Repository System implementation is continually being enhanced in order to extend functionality and improve usability issues. The usability feedback is obtained through modeling activities by members of the development team, who are also working to enhance the content of the example product repositories available in the database.

Ongoing efforts in related projects at NIST are working to develop extensions to the Core Product Model that provides the representational foundation for this project. Extensions in progress include the representation of complex assembly models, process planning information, and design rationale information. As these extensions are completed, additional work will be necessary to extend the modeling capabilities of the Design Repository System accordingly.

The most recent addition to the Design Repository System tool suite is a search tool, which enables users to perform searches that can be of use in early stages of product development. For example, a user can search the design repository database for components not only based on their name (as might be possible in a typical commercial CAD system), but also based on their function. Although the search tool is currently functional, improvements in this preliminary prototype are anticipated to improve both functionality and usability.

ACKNOWLEDGMENTS

The author would like to thank Khanh Trieu and Guilhem Assant, whose technical reports and implementation documentation provided much of the implementational detail presented in this paper.

REFERENCES

1. Gorti, S. R., A. Gupta, G. J. Kim, R. D. Sriram and A. Wong (1998), "An Object-Oriented Representation for Product and Design Process", *Computer-Aided Design*, Vol. 30, No. 7, pp. 489-501.
2. Szykman, S., R. D. Sriram, and S. J. Smith (Eds.) (1998), *Proceedings of the NIST Design Repository Workshop*, NISTIR 6159, National Institute of Standards and Technology, Gaithersburg, MD, November, 1996.
3. Shooter, S. B., W. Keirouz, S. Szykman and S. J. Fenves (2001), "A Model of the Flow of Design Information in Product Development," *Engineering With Computers*, Vol. 16, No. 3-4, pp. 178-194.
4. Szykman, S., S. J. Fenves, S. B. Shooter and W. Keirouz (2001), "A Foundation for Interoperability in Next-generation Product Development Systems," *Computer-Aided Design*, Vol. 33, No. 7, pp. 545-559.
5. Fenves, S. J. (2001), *A Core Product Model for Representing Design Information*, NISTIR 6736, National Institute of Standards and Technology, Gaithersburg, MD, April.
6. Hardwick, M. and D. Loffredo (1995), "Using EXPRESS to Implement Concurrent Engineering Databases," *Proceedings of the 1995 ASME Computers in Engineering Conference and Engineering Database Symposium*, Boston, MA, September, pp. 1069-1083.

7. Kim, T. S., S.-H. Han, and Y. J. Shin (1996), "Product Data Management Using AP203 of STEP Standard," *Proceedings of the 1996 ASME Design Engineering Technical Conferences and Computers in Engineering Conference*, Paper No. 96-DETC/DAC-1069, Irvine, CA, August.
8. Shah, J. J., D. K. Jeon, S. D. Urban, P. Bliznakov, and M. Rogers (1996), "Database Infrastructure for Supporting Engineering Design Histories," *Computer-Aided Design*, Vol. 28, No. 5., pp. 347-360.
9. Wood III, W. H. and A. M. Agogino (1996), "Case-Based Conceptual Design Information Server for Concurrent Engineering," *Computer-Aided Design*, Vol. 28, No. 5, pp. 361-370.
10. Bilgic, T. and D. Rock (1997), "Product Data Management Systems: State-of-the-Art and the Future," *Proceedings of the 1997 ASME Design Engineering Technical Conferences*, Paper No. DETC97/EIM-3720, Sacramento, CA, September.
11. Wong A. and R. D. Sriram (1993), "SHARED: An Information Model for Cooperative Product Development," *Research in Engineering Design*, Vol. 5, No. 1, pp. 21-39.
12. de Kleer, J. and J. S. Brown (1983, "Assumptions and Ambiguities in Mechanistic Mental Models," *Mental Models*, D. Gentner and A. L. Stevens (Eds.), Lawrence Erlbaum Associates, New Jersey, pp. 155-190.
13. Iwasaki Y. and B. Chandrasekaran (1992), "Design Verification through Function and Behavior-Oriented Representations: Bringing the Gap between Function and Behavior," *Artificial Intelligence in Design '92*, J.S. Gero (Ed.), Kluwer Academic Publishers, Boston, pp. 597-616.
14. Chandrasekaran, B., A. Goel, and Y. Iwasaki (1993), "Functional Representation as Design Rationale," *IEEE Computer*, January, pp. 48-56.
15. Goel, A., S. Bhatta, and E. Stroulia, (1996), "KRITIK: An Early Case-Based Design System," *Issues and Applications of Case-Based Reasoning to Design*, M. Maher and P. Pu (Eds.), Lawrence Erlbaum Associates, New Jersey.
16. Goel, A., A. Gomez, N. Grue, J. W. Murdock, M. Recker, and T. Govindaraj (1996), "Explanatory Interface in Interactive Design Environments," *Artificial Intelligence in Design '96*, J. S. Gero (Ed.), Kluwer Academic Publishers, Boston.
17. Alberts L.K. and F. Dikker (1992), "Integrating Standards and Synthesis Knowledge Using the YMIR Ontology," *Artificial Intelligence in Design '94*, J.S. Gero and F. Sudweeks (Eds.), Kluwer Academic Publishers, Boston, pp. 517-534.
18. Henson, B., N. Juster and A. de Pennington (1994), "Towards an Integrated Representation of Function, Behavior and Form," Computer Aided Conceptual Design, Proceedings of the 1994 Lancaster International Workshop on Engineering Design, Sharpe J. and V. Oh (eds.), Lancaster University EDC, Lancaster, pp. 95-111.
19. Ranta, M., M. Mäntylä, Y. Umeda and T. Tomiyama (1996), "Integration of Functional and Feature-Based Product Modelling – the IMS/GNOSIS Experience," *Computer-Aided Design*, Vol. 28, No. 5, pp. 371-381.
20. Umeda, Y., M. Ishii, M. Yoshioka, Y. Shimomura and T. Tomiyama (1996), "Supporting Conceptual Design Based on the Function-Behavior-State Modeler," Artificial Intelligence for Engineering Design, Analysis and Manufacturing. Vol. 10, pp. 275-288.
21. Shimomura, Y., M. Yoshioka, H. Takeda, Y. Umeda and T. Tomiyama (1998), "Representation of Design Object Based on the Functional Evolution Process Model," *ASME Journal of Mechanical Design*, Vol. 120, No. 2, pp. 221-229.
22. Qian L. and J. S. Gero (1996), "Function-Behavior-Structure Paths and Their Role in Analogy-Based Design," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, Vol. 10, No. 4, pp. 289-312.
23. Szykman, S., J. W. Racz and R. D. Sriram (1999), "The Representation of Function in Computer-based Design," *Proceedings of the 1999 ASME Design Engineering Technical Conferences and Computers in Engineering Conference*, Paper No. DETC99/DTM-8742, Las Vegas, NV, September.
24. Karne, R. K., S. V. Dandekar, S. Poluri, G. Chen, J. S. Baras, D. S. Nau, M. O. Ball, E. Lin, V. S. Trichur and J. T. Williams (1998) "Web-It-Man: A Web-based Integrated Tool for Manufacturing Environment," *Proceedings of the 1998 ASME Design Engineering Technical Conferences and Computers in Engineering Conference*, Paper No. DETC98/CIE-5524, Atlanta, GA, September.
25. Xiao, A., H.-J. Choi, R. Kulkarni, J. K. Allen, D. Rosen, F. Mistree and S. C. Feng (2001), "A Web-based Distributed Product Realization Environment," *Proceedings of the 2001 ASME Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, Paper No. DETC2001/CIE-21766, Pittsburgh, PA, September.
26. Pahng, G.-D. F., S. Bae and D. Wallace (1998), "Web-based Collaborative Design Modeling and Decision Support," *Proceedings of the 1998 ASME Design Engineering Technical Conferences and Computers in Engineering Conference*, Paper No. DETC98/EIM-5681, Atlanta, GA, September.
27. Sistla, R., A. R. Dovi and P. Su (2000), "A Distributed, Heterogeneous Computing Environment for Multidisciplinary Design and Analysis of Aerospace Vehicles," *Advances in Engineering Software*, Vol. 31, No. 8-9, pp. 707-716.
28. Jayaram, U., S. Jayaram, Y. Yang and K. Lyons (2000), "CORBA-based Collaboration in a Virtual Assembly Design Environment," *Proceedings of the 2000 ASME Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, Paper No. DETC2000/CIE-14585, Baltimore, MD, September.
29. Liang, J., J. J. Shah, R. D. Souza, S. D. Urban, K. Ayyaswamy, E. Harter and T. Bluhm (1999), "Synthesis of Consoludated Data Schema for Enineering analysis from Mul-

tiple STEP Application Protocols," *Computer-aided Design*, Vol. 31, No. 7, pp. 429-447.
30. Zhang, F. and D. Xue (2002), "Distributed Database and Knowledge Bas Modeling for Concurrent Design," *Computer-Aided Design*, Vol. 34, No. 1, pp. 27-40.
31. Rajagopalan, S., J. M. Pinilla, P. Losleben, Q. Tian and S. K. Gupta (1988), "Integrated Design and Rapid Manufacturing over the Internet," *Proceedings of the 1998 ASME Design Engineering Technical Conferences and Computers in Engineering Conference*, Paper No. DETC98/CIE-5519, Atlanta, GA, September.
32. Jin, Y., and W. Zhou (1999), "Agent-based Knowledge Management for Collaborative Engineering," *Proceedings of the 1999 ASME Design Engineering Technical Conferences and Computers in Engineering Conference*, Paper No. DETC99/EIM-9022, Las Vegas, NV, September.
33. Chadha, B. and J. Welsh (2001), "An Architecture for Virtual Prototyping of complex Systems," *Proceedings of the 2001 ASME Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, Paper No. DETC2001/CIE-21239, Pittsburgh, PA, September.