Web-based System for Design Artifact Modeling

Simon Szykman, Janusz Racz, Ram D. Sriram, Christophe Bochenek*
Manufacturing System Integration Division
National Institute of Standards and Technology
Building 304, Room 6
Gaithersburg, MD 20899, USA

December 1, 1998

Abstract

Engineering product development in today's industry is becoming increasingly knowledge intensive. The NIST Design Repository Project, at the National Institute of Standards and Technology, is working to develop infrastructural technologies to support the use of design repositories in industry. In contrast to traditional design databases, design repositories more actively support knowledge-based design, serving not only as archives, but as repositories of heterogeneous information that are designed to enable representation, capture, sharing, and reuse of corporate design knowledge. This paper presents a language that has been developed for the modeling of engineering design artifacts. The implementation of a prototype tool suite, which includes intelligent web-based interfaces that allow distributed users to create, edit and browse design repositories, is also presented.

1 Introduction

Design of complex engineering systems is increasingly becoming a collaborative task among designers or design teams that are physically, geographically, and temporally distributed. The complexity of modern products means that a single designer or design team can no longer manage the complete product development effort. Developing products without sufficient expertise in a broad set of disciplines can result in extended product development cycles, higher development costs, and quality problems. On the other hand, ensuring comprehensive technical proficiency in a world where trends are toward more multidisciplinary design can become a costly undertaking for a company.

Driven by such issues, companies are increasingly staffing only their core competencies in-house and depending on other firms to provide the complementary design knowledge and design effort needed for a complete product. Designers are no longer merely exchanging geometric data, but more general knowledge about design and design process, including specifications, design rules, constraints, rationale, etc. As design becomes increasingly knowledge-intensive and collaborative, the need for computational design frameworks to support the representation and use of knowledge among distributed designers becomes more critical. Due to the explosive growth of the Internet and associated information infrastructure, as well as the ubiquity of World Wide Web browsers, the use of the Internet and the World Wide Web as methods for communication and information transfer is increasing.

In addition to sharing and exchanging information, pressure to reduce product development times has resulted in an increased focus on methods for representing and storing engineering artifact knowledge in a way that facilitates its retrieval and subsequent reuse. Merely providing access to schematics, computer-aided design (CAD) models of artifacts and written documentation is inadequate for this purpose. The emerging research area of design repositories is aimed at addressing these industry needs.

*e-mails: `szykman@cme.nist.gov`, `jwracz@cme.nist.gov`, `sriram@cme.nist.gov`, `bchrist@cme.nist.gov`

A design repository is an intelligent knowledge-based design modeling system used to facilitate the representation, capture, sharing, and reuse (search and retrieval) of corporate design knowledge. It should be noted that although the term design repositories has not yet found its way into daily usage in industry, many companies are migrating from traditional design databases to design repositories[1]. Design repositories are distinguished from traditional design databases in a number of significant ways:

- Traditional design databases are typically more data-centric than knowledge-centric, while design repositories attempt to capture a more complete design representation that may include characterization of function, behavior, design rules, simulation models, and so on.
- Design databases are generally more homogeneous in the kinds of information they contain (e.g. images, CAD models, and unstructured text/documentation). Design repositories may include, in addition to these, formal data/information models, structured text (specialized languages for representing function, design rules, logical expressions), mathematical simulation models and more.
- Design databases tend to be static sources of information (though their contents may grow with time). Design repositories are designed not only for storage of information, but to support retrieval and reuse of design data in more sophisticated ways such as search for components/assemblies that satisfy required functions, explicit representation of physical and functional decompositions and the mappings between them, etc.

The NIST (National Institute of Standards and Technology) Design Repository Project involves research toward creating a technical foundation for the development of design repositories, addressing a broad set of issues from knowledge representation and information models, to taxonomies of design function, to interfaces. The project was motivated by needs identified at an industry workshop held at NIST in November 1996 [25] and, as such, focuses on needs determined by industry participants to be important to future knowledge-based design systems.

This paper presents a summary of the knowledge representations used within the NIST Design Repository Project artifact modeling system, followed by details about the implementation and intelligent web-based interfaces for creating, editing and browsing design repositories. A more general summary of the project and associated research issues can be found in [26]. Research done in areas related to the work described in this paper are summarized in Section 2. Section 3 describes the knowledge representation used for modeling design artifacts. The implementation, system architecture and interfaces are presented in Section 4. Section 5 discusses conclusions and future directions for this project.

2 Related Work

Traditional CAD systems are limited to representation of geometric data and other types of information relating to geometry that may include 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 primarily on database-related issues and do not place a primary emphasis on information models for artifact representation (e.g. [3], [11], [17], [23], [28]).

Furthermore, although these systems can represent some kinds of non-geometric knowledge (e.g. information about the design process, manufacturing process, bills of materials, 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,

[1] Just as the word "database" can refer to either a database management system or an individual information store and its content, in this paper, the term "design repository" will be used to describe both the modeling system (underlying representation, interfaces and mechanisms) as well as an specific design artifact model and its content. The intended meaning in a particular instance should be clear from the surrounding context.

behavior and structure has been identified by Bilgic and Rock in [2] as a shortcoming of existing PDM systems.

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

The artifact modeling representation used in this research builds on earlier work in developing an object-oriented representation format that provides a high level division into form, function, and behavior ([27], [10]). The division of design artifact knowledge into these categories has its roots in earlier work in intelligent design system development. Examples of work in this area includes the qualitative simulation work in [7], behavioral and functional representation in [12], functional representation in [4] and successive representation from projects such as KRITIK [8] and INTERACTIVE KRITIK [9], the YMIR project [1], CONGEN [10], and others.

3 Knowledge Representation

Design repositories are intended to support the storage, retrieval and reuse of engineering knowledge about artifacts that have been designed. This knowledge includes the geometric description such as drawings and/or CAD models, as well as complementary knowledge concerning characterization of artifacts, function, behavior, relationships and interconnections between them. Information is represented using a formal knowledge representation that can be comprehended by humans, but which, unlike natural language, can be interpreted by computers and used for (partially) automated reasoning.

In order to achieve this objective, an artifact modeling language has been developed that consists of a *Data Language* (DL) and a *Design Representation Language* (DRL)[21]. The Data Language describes the syntax and data structures for a highly generic object-oriented (object/class) paradigm that can be applied to any of many different domains; the Design Representation Language combines the Data Language with an engineering context to provide the means for modeling design artifacts. The artifact modeling language is derived from the SHARED object model [27], which is implemented within a conceptual design shell called CONGEN (an object-oriented domain-independent design support framework) [10].

3.1 The Data Language

The Data Language (DL) developed in this research is based on the SHARED object model[27]. This model, which is useful not only for efficient data handling, can be easily adopted to automate a design process. For databases, which are destined to be used on a large scale in a distributed environment (e.g. Internet), the SHARED model was modified to address rigorous requirements in arrangement of storage, indexing, searching or concurrent access.

The DL consists of four basic types of entities: objects[2], relationships, and (object) classes and relationship classes from which objects and relationships are instantiated. Each of these entity types are identified by a unique name.

3.1.1 Objects

An object $\mathbf{o}$ is defined as $\mathbf{o}(\mathbf{o}_{id},\mathbf{c}_{id},\mathbf{A},\mathbf{R})$, where $\mathbf{o}_{id}$ is a unique object identifier (i.e. a name), $\mathbf{c}_{id}$ identifies the class the object belongs to (i.e. the *parent* of the object), $\mathbf{A}$ is a set of attributes represented by names

[2] In its general usage, the word "object" is usually used to describe any of numerous types of entities in an object-oriented system. In this work, however, the term refers to a specific kind of data item, and thus relationships are not referred to as "objects" though they could be considered objects in the generic sense. Because of this distinction, the term "entity" is used to refer to the four types of data items.

of attributes and their values (which may be strings, numbers, references to other entities, etc.), and **R** is a set of relationships between sets of objects.

An example of an object data structure is shown in Fig. 1.[3] The object shown defines the artifact[4] named *drill_artifact_1* (its o_{id}), and *Drill_artifact*, identifying it as an instance of that class. Its five attributes are *function*, *form*, *behavior*, *full_name*, and *description*. The values of the first three attributes are in brackets, indicating that they are references to other entities having those names; the other two attributes have strings as values. The list of relationships for the *drill_artifact_1* object consists of a single relationship, again indicated by an entity name in brackets.

```
Object:
parent:  [Drill_artifact]
oid:     [drill_artifact_1]
A:       function [drill_artifact_1_Function];
         form [drill_artifact_1_Form];
         behavior [drill_artifact_1_Behavior];
         full_name ``Black & Decker VP840 Drill'';
         description ``Battery powered power drill/screwdriver'';
R:       [drill_artifact_1_Has_part];
```

Figure 1: Example schema for an object entity

3.1.2 Relationships

Relationships among objects are represented explicitly in the Data Language. This aids in organizing groups of objects (in hierarchical or tree structures, for example), as well as to support human or automated navigation among collections of objects, which in our case are design repositories. The generic relationship $\mathbf{r}$ is defined similarly to the object: $\mathbf{r}(\mathbf{r}_{id},\ \mathbf{r}_{cid},\ \mathbf{RO},\ \mathbf{A})$, where $\mathbf{r}_{id}$ is a unique identifier (name) of the relationship, **RO** is a set of roles that describe the nature of the relationship between the associated entities, and **A** is a set of attributes defined the same way as for an object.

An example of a relationship entity is shown in Fig. 2. The entity is an instance of the *Has_part* relationship class, and defines the decomposition of a *composite* assembly (the `drill_artifact_1` object) into a set of *components* which may be either subassemblies and/or individual components. This relationship is bi-directional; it references both the *composite* and the *components*, and it is also referenced in the data structures for both the *composite* (as seen in Fig. 1) and in the *components* (not shown). Such an approach enables the organization of a collection of objects using relationships in either a bottom up or a top down fashion, or a mix of both. The example shown is taken from the artifact modeling domain, and thus the *composite* and *components* roles apply to the representation of artifacts. These roles are associated with the Design Representation Language, which has roles specific to artifact modeling; the Data Language itself does not restrict the roles associated with a relationship to a given domain.

3.1.3 Object Classes

The structures of individual objects and their syntax are described by object classes (heretofore referred to simply as classes). A class $\mathbf{c}$ is defined as $\mathbf{c}(\mathbf{c}_{id},\ \mathbf{s}_{cid},\ \mathbf{A}_D,\ \mathbf{R}_c)$, where $\mathbf{c}_{id}$ is a unique class identifier, (name) $\mathbf{s}_{cid}$ is a superclass (i.e. a parent class) of which $\mathbf{c}$ is a subclass, $\mathbf{A}_d$ is a set of attribute definitions, and $\mathbf{R}_c$

[3]Recall that as described previously, the Data Language is generic and can be applied to any of many domains. Although the example shown in the figure has attributes and values that associat the object with an engineering artifact, there is nothing within the generic object schema that restricts it to this domain. The basic object schema could just as easily be used to represent an employee in a corporate personnel database.

[4]Artifacts are a type of object and will be described in greater detail in the section on the Design Representation Language.

```
Relationship:
parent:         [Has_part]
rid:            [drill_artifact_1_Has_part]
RO:             composite [drill_artifact_1];
                components
                  {[housing_system_1] [power_system_1] [control_system_1]
                   [drill_system_1] [white_wire_1] [red_wire_1]
                   [black_wire_1] [black_wire_3]};
A:
```

Figure 2: Example schema for a relationship entity

is a set of relationship classes. An object **o** belonging to a class **c** contains the attributes and relationships of that class. This allows generic class to define the schema for instantiated objects belonging to that class. An object belonging to a class generally has the slots defined by the class, but may also differ somewhat, either by having additional slots not specified in the class definition, or by allowing unnecessary slots defined in the class definition to remain unused (the unused slots would have no values). This method of relating objects to classes differs from most object-oriented programming formulations, but is consistent with the intended use of the Data Language. Specifically, the classes are intended to define useful schemata while building in representational flexibility that is not required in more traditional object-oriented programming environments.

Fig. 3 depicts the class named `Tool_artifact`. It is descendant of the class `Artifact`, which means, that it inherits properties predefined in its parent's definition. A class can possess additional attribute definitions or relationship classes, which are not possessed by its ancestor and may not use all of them. The drill artifact shown in Fig. 1 is an instance of `Drill_artifact`, which is a subclass of `Tool_artifact`, which is a subclass of `Artifact`. The slots *function*, *form*, and *behavior* which appear in Fig. 1 are inherited from the `Artifact` class, and therefore are not shown explicitly in the `Tool_artifact` class definition shown in Fig. 3. The `Tool_artifact` class definition adds a new attribute called *full_name*, which does not appear in the `Artifact` class definition.

```
Class:
parent:   [Artifact]
cid:      [Tool_artifact]
A:        STRING: full_name;
R:        [Has_part];
```

Figure 3: Schema for the a class entity

3.1.4 Relationship Classes

Relationship classes refer to relationships and play the same role as classes do for objects. A relationship class **r** is defined as $\mathbf{r}_c(\mathbf{r}_{cid}, \mathbf{s}_{rcid}, \mathbf{RO}_d, \mathbf{A}_d)$, where $\mathbf{r}_{cid}$ is a relationship class identifier (name), $\mathbf{s}_{rcid}$ is a relationship superclass (or parent class), $\mathbf{RO}_d$ is a set of role definitions, and A_d is a set of attribute definitions.

The relationship class showed in Fig. 4 describes the schema of the `Has_part` relationship. Note that the curly braces that appear to the left of the *components* role indicate a list of entities, not a single entity. Thus, a `Has_part` relationship defines the decomposition of a single *composite*, which must belong to the

Artifact class, into a set of *components* which must also belong to the Artifact class. No attributes are defined for this class.

```
RelationshipClass:
parent:              NULL
rcid:                [Has_part]
RO:                  [Artifact] composite;
                     {[Artifact]} components;
A:
```

Figure 4: Example of a relationship class entity

Both the (object) class and relationship class hierarchies contain top level entities, which have no parents. The Has_part is an example of such a relationship class, and therefore the value of its parent slot is NULL. These special entities are part of the Design Representation Language (DRL) and are described in the next subsection.

3.2 The Design Representation Language

The previous subsection has defined the basic schemata and data structures of the Data Language in isolation from any engineering context (though the examples did include such context). The engineering context is defined through the Design Representation Language (DRL), and includes the semantics and class hierarchies of *artifacts*, taxonomies of *functions* and *flows*, the concept of compositional, functional and behavioral decomposition, and many engineering-specific attributes.

In the DRL an artifact $\mathbf{a}$ is defined (via its class definition) as an object whose subset of attributes $\mathbf{A}_a$ contains references to objects belonging to three other fundamental classes: *form* - the geometry or physical properties (material, color, surface finish, etc) of the artifact, *function* - an indication of the purpose of an artifact, and *behavior* - a casual account of the operation of the artifact.

Artifacts are used to represent physical assemblies, subassemblies and components; these are linked to each other by the bi-directional Has_part relationship to define the compositional (physical) hierarchy of the product. Similarly, a Has_function relationship is used to define a functional decomposition of a product. In addition to any attributes mentioned above, artifacts may contain additional attributes specific to individual objects. The set of classes that are descended (through *parent* slots) from the Artifact class create a hierarchy that describes a conceptual classification of products, which is independent of their assembly structures. Using the DRL, any artifact is an instance of some type of Artifact class.

Since more and more CAD systems have the ability to import and export STEP (Standard for the Exchange of Product Model Data [14]) data, this format is used as the basis for representation of geometry in this project. Thus in the DRL, objects instantiated from the *form* class have as attributes pointers to data files in STEP format, specifically STEP AP (Application Protocol) 203 [15]. In addition, to facilitate remote visualization of artifacts, geometry is also accessible in the Virtual Reality Modeling Language (VRML) [16]) format.

Artifact behavior specifies the response of an artifact to input condition or behavioral states. The Behavior class contains attributes that refer to a set of *input* and *output* objects, as well as relationships to decompose behavior into subbehaviors (using *Has_subbehavior* and *Accomplished_by* relationships) to encode the link between the behavior of an artifact and the behavior of another artifact, that directly accomplishes that goal. The representation scheme for behavioral information within this work is not yet mature and will be developed further as subsequent research.

In the DRL, *functions* describe interactions between *flows*[5], which are treated as inputs and outputs

[5]The term "flow" in this work is used in a sense similar to that proposed by Pahl and Beitz [22], referring to the flow of materials, energy and information.

to functions. Like artifacts, functions are represented using objects which have a schema that defines the function inputs and outputs as references to other entities (objects) representing flows, which in turn have their own schemata. Objects representing functions are instantiated from classes that are descended from the `Function` class.

Functions are decomposed using two type of relationships: `Has_subfunction`, which decomposes a complex function into simpler ones, and `Satisfied_by` playing the same role as `Accomplished_by` does for behavior. The first type of relationship enables the definition of a functional decomposition of a design, while the latter is used to avoid multiple identical objects, thereby reducing the overall number of objects required to describe a product. Because artifacts refer to functions, functions refer to flows associated with those functions, and flows refer to artifacts associated with those flows, the DRL provides a mapping between the representations of the physical decomposition and the functional decomposition of a product.

The use of functional decomposition in design simplifies the design process and facilitates comprehension of the fundamental description of a product. Eventually, the process of subdivision reaches a level at which further decomposition is not beneficial. The `Function` taxonomy can provide the natural set of leaves[6], which terminate this tree-like decomposition. The problem is to provide a terminology which is domain-independent and generic enough to model a broad variety of engineering artifacts, and yet small enough to provide a manageable standardized vocabulary and to facilitate indexing and retrieval for design reuse. To accomplish these conflicting requirements a the development of comprehensive taxonomies of functions and associated flows is in progress.[7] For illustrative purposes, Fig. 5 shows an abridged portion of the `Function` and `flow` taxonomies (i.e. class hierarchies) used in this work.

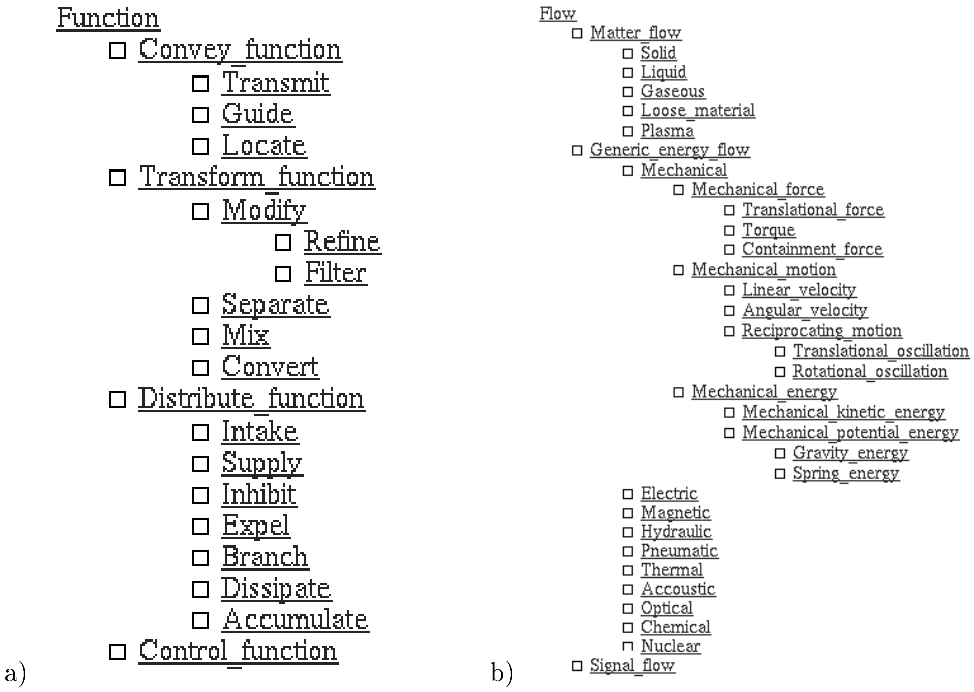


Figure 5: Class hierarchies: a) *Function* taxonomy b) *Flow* taxonomy

The first level of the function hierarchy consists of four basic function classes that are subclasses if the root class `Function`. These are: `Convey`, `Transform`, `Distribute` and `Control` (see Fig. 5a). If a more precise description of elementary functions is required, classes from lower levels of the function class hierarchy may be used. Although the function taxonomy has currently been expanded only to two levels in depth,

[6]By "leaf" we mean an *instance* of a *Function* class, expressing a basic interaction between flows.
[7]Other work in the area of function representation is being studied, such as [18], [19], [24].

this set of function classes allows the representation of most artifacts in the domain of mechanical systems design. The branches of this tree can be further expanded in cases when the existing taxonomy lacks an appropriate term to describe a function required for modeling a specific artifact or component. As seen in Fig. 5b, the taxonomy of flows follows the Pahl and Beitz model with the root class `Flow` being decomposed into three basic subclasses: `Matter`, `Generic_energy` and *Signal*.

4 Implementation

A primary objective of the NIST Design Repository Project is to develop a computational framework for the creation of design repositories, and a proof-of concept prototype to demonstrate their benefits. This research has resulted in the implementation of a suite of tools for distributed development of, and access to, design repositories. The system that has been implemented includes web-based interfaces that enable a designer to create, edit, and browse design repositories, accessed via the Internet by multiple distributed clients using common web browsers.

Two types of interfaces have been implemented. The first is a Design Repository Editor used to create and update design repositories. This editor interacts with an ASCII text file containing the information to be stored in a repository. The format of this file follows the schemata defined by DL and the DRL; the contents of the formatted text file for a power drill design repository include the data structures used as examples in the previous section. The second interface is the Design Repository Browser[8] that retrieves design repository information from a database management system (DBMS), in which it is stored. The use of this database provides capabilities such as transaction management, consistency maintenance, distributed databases, synchronization of mirrored databases, information caching, etc. The schemata for the database representation of a design repository are unchanged, but within the DBMS information is stored in a compiled form rather than in a formatted text file.

The Design Repository tool suite includes several components in addition to the web-based interfaces:

- ObjectStore^{TM}[9], a commercial object-oriented database management system, developed by Object Design, Inc.
- A Design Repository Compiler which takes a formatted text file created by the Design Repository Editor and transfers the contents to an ObjectStore^{TM} database.
- An information extractor or "decompiler" which takes the contents of a database and replicates them in a formatted text file for further editing.
- STEP/Works, a STEP AP 203 viewer developed by International TechneGroup, Inc. for local (non web-based) visualization of STEP-based geometry, desirable in some cases since VRML provides a less comprehensive representation of geometry.

4.1 Architecture

Since the interfaces are web-based, no special client software is required other than a web browser. The architecture of the Design Repository Browser is illustrated in Fig. 6 As the designer uses the point-and-click interface, the browser sends Hypertext Transfer Protocol (HTTP) requests to an HTTP server. Common Gateway Interface (CGI) scripts running on the HTTP server parse and process those requests into database queries that are sent to a database server running on a different machine. The database server queries a

[8]Not to be confused with a web browser, the Design Repository Browser is the user interface to a design repository that allows the designer to navigate through an artifact representation. The Design Repository Browser is a web-based interface that runs within a web browser.

[9]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 National Institute of Standards and Technology.

design repository stored in an ObjectStore^{TM} database. The database queries retrieve information from the repository and return the data as text to the CGI scripts, which then format the data in HTML for viewing through a web browser. The formatted data is then sent from the HTTP server back to the user. In addition to data about the artifacts themselves, the repositories also include Universal Resource Locators (URLs) for VRML models of the various assemblies and components. When the user follows one of these links, the model is retrieved directly using its URL without requiring calls to the database server, and is displayed for the user using a VRML plug-in or viewer.

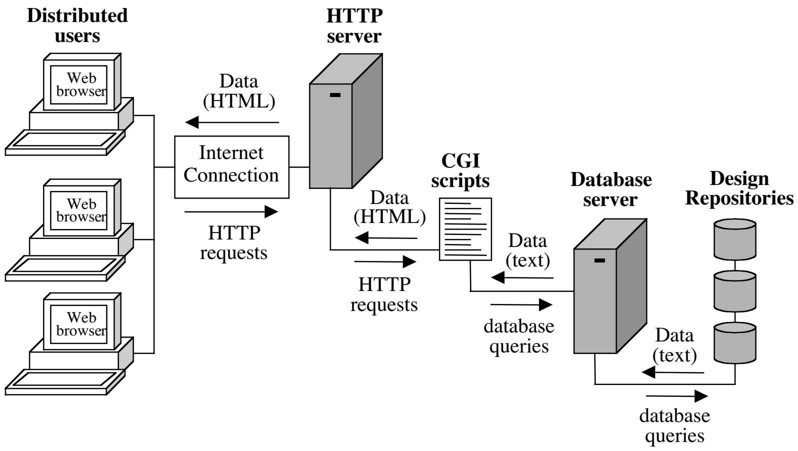


Figure 6: Architecture of the Design Repository Browser

Interactions involving the Design Repository Editor interface are quite similar. The main difference is the introduction of a batch processing mode designed as a time-saving feature. When creating or editing a design repository, the contents of a repository are also stored in a specially formatted text file as described above. As a repository is changed or modified, the text file is updated accordingly without updating the databases files themselves. Artifacts modeled in design repositories can consist of many entities; for a point of reference, a power drill repository that has been developed contains over 250 entities. Not updating the database files each time an entity is created or modified eliminates many time-consuming calls to the database server, thereby saving a considerable amount of time. Once an editing session is complete, the user compiles the text file into a database using the Design Repository Compiler that transfers the entire contents of this formatted text file into a database in a batch operation.

The development of a new version of the system architecture has recently begun. The motivation and a more detailed description of the new architecture are discussed in Section 5 of the paper.

4.2 Web-based Design Repository Editor

A formatted text file that follows the schemata for the data structures defined by the DL and DRL, can be created using any text editor. But this is a very complicated and time consuming task, which requires a designer to be familiar with specific syntax of the design repository entities, to have detailed knowledge about the artifact class hierarchies, the function and flow taxonomies, and to expend considerable effort to maintain consistency within the data structures. To simplify the design repository development process an intelligent web-based Design Repository Editor, which can be used via most common web browsers (e.g. Netscape Navigator or Microsoft Internet Explorer) has been implemented. This interface allows the user to define new classes, to create an artifact model by creating objects and relationships from existing classes, and to describe physical, functional and behavioral decompositions as well as the mappings between them. Fig. 7 shows the Design Repository Editor being used for two different actions: the editing of the artifact class hierarchy (Fig. 7a) and the creation of a new function object (Fig. 7b).

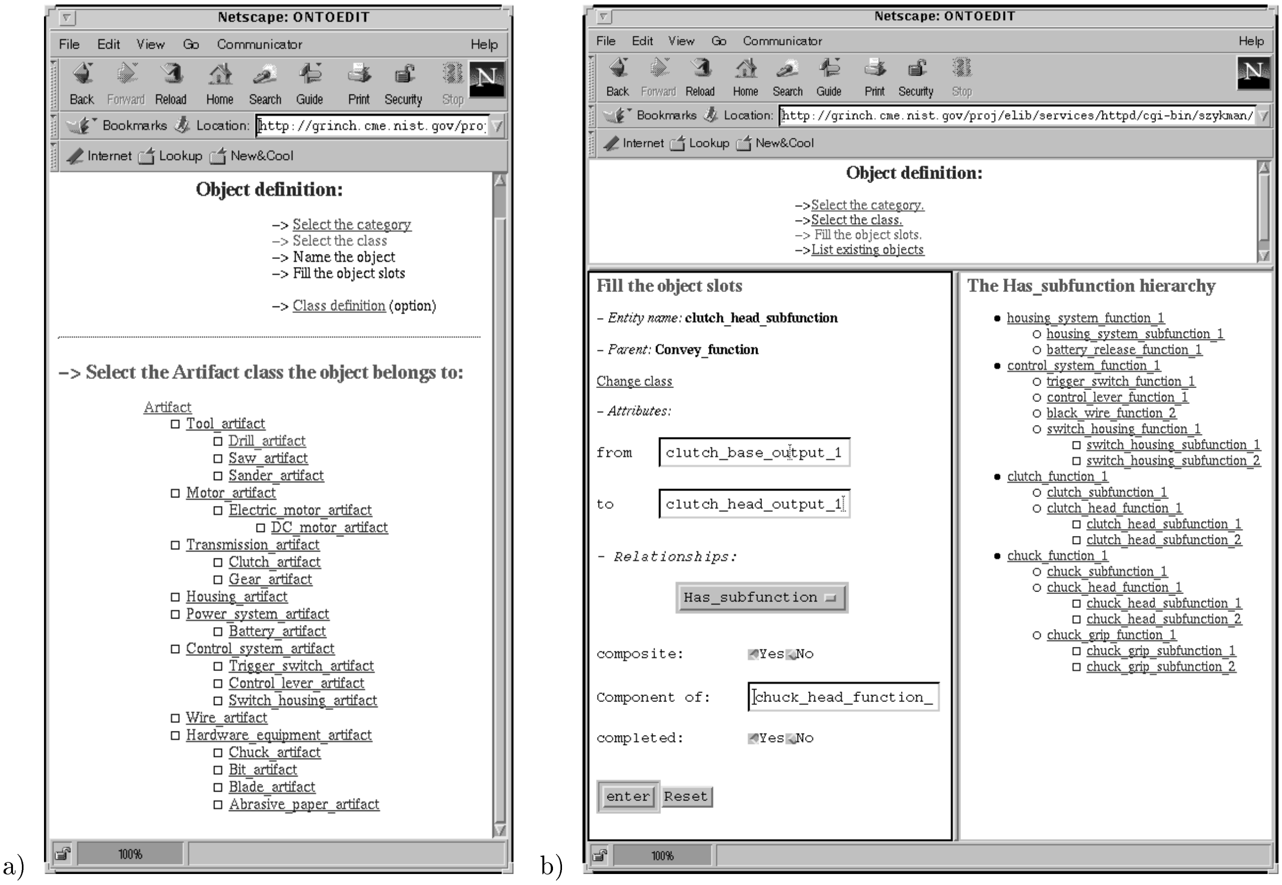


Figure 7: Web based CGI editor: a) the window to assign the class, b) the multiframe window for completing the object form and/or for creating the object hierarchy

All web pages created by the Design Repository Editor provide a point-and-click interface that allows the user to navigate across the stored data and access the editor functionality. According to DRL, the key entities of a design repository are objects (used to represent artifacts, functions, flows, behaviors, etc) and relationships. The creation of a new function as is shown in Fig. 7b requires only three simple actions:

1. Select a class from which the function should be instantiated, which can be accomplished by clicking on the appropriate item in the function class hierarchy, shown in the right-hand portion of the window in Fig. 7b.

2. Name the object.

3. Fill in the slots for the attributes and relationships as required.

The Design Repository Editor is an intelligent interface that removes much of the burden from the designer. These capabilities include:

- Automatic schema generation. When an object is instantiated from a class, a form is automatically generated with the appropriate slots based on the schema defined by the class description without requiring the designer to know *a priori* all the slots needed for a class.

- Automatic entity linking. When a link to an existing entity is specified by the designer, the data structures are automatically updated to reflect those links. To illustrate, when the designer submits the form shown in Fig. 7b, the current repository will be searched for each of the entities that appear in the form (e.g. `clutch_base_output_1`) and will create a link to those entities if they exist.
- Automatic entity creation and to-do list. When a link to an entity is specified and that entity does not exist, a "dummy" entity is created and named with the appropriate name. The editor maintains a to-do list, which is a list of all items that were created because they didn't exist when links to them were specified, but that the designer still has to go back to and fill in the required attributes and values. At any point, the designer can click on an entity name on that list, and a form is automatically generated with the appropriate slots based on the schema defined by the entity's class.
- Partial entity creation. As can be seen at the bottom of Fig. 7b, the designer can check a box indicating that an entity that has been created is not yet complete. This will result in that entity being added to the to-do list so that the designer does not forget to return and finish filling out the required information.
- Automatic relationship entity creation, naming and linking. When, for example, an artifact called `Drill_artifact_1` is created and the designer specifies that it has a `Has_part` relationship, all the designer must do is indicate the *components* into which that artifact is decomposed. The entity for that `Has_part` relationship is automatically created, named `Drill_artifact_1_has_part`, and linked to the `Drill_artifact_1` object (see Fig. 2 for reference). As before, if objects corresponding to those *components* already exist, they too are linked to the relationship entity. If not, "dummy" objects are again created and added to the to-do list. In this manner, the designer can specify physical, functional and behavioral hierarchies without ever having to explicitly create relationship entities; it is done automatically by the Design Repository Editor.

The intelligent Design Repository Editor provides the designer with several significant advantages. First, it encompasses a great deal of knowledge about the engineering context, which often translate into constraints on how an artifact can be modeled, so that the designer is not required to remember all these constraints. Second, by maintaining a list of dummy objects, the designer is not restricted to modeling an artifact in a specific manner, but can create a model using a top down approach, a bottom up approach, or a combination of the two moving back and forth between levels of detail. Finally, not only does the automatic entity creation, naming and linking save time for the designer, it aids tremendously in consistency maintenance by eliminating a significant portion of the errors that would occur when manually creating a design repository using a text editor (e.g. misspelling an entity name causing a broken link, or specifying a link to an object but forgetting to create it).

4.3 Web-based Design Repository Browser

The other interface developed within the framework of this research is the Design Repository Browser, which is the user interface to a design repository that allows the designer to navigate through an artifact representation. The Design Repository Browser extracts information about entities from a design repository database where they are stored in object-oriented data structures, and provides this information to the user.

Fig. 8 shows two different images of the Design Repository Browser. The multiframe window (Fig. 8a) created by the Design Repository Browser enables the user to navigate across an existing design repository. Such a window is accessed via most web browsers and consists of four frames, one static (the upper frame which links to information about the NIST Design Repository Project) and three dynamic ones below that one.

In the three dynamic frames, each of the underlined names is a hyperlink to a CGI script which retrieves information from a design repository database and updates the browser. The bottom left frame presents information about an entity in the design repository (this is the same information that is shown in the text-based data structure shown in Fig. 1). The user can browse a repository by clicking any of the links in that

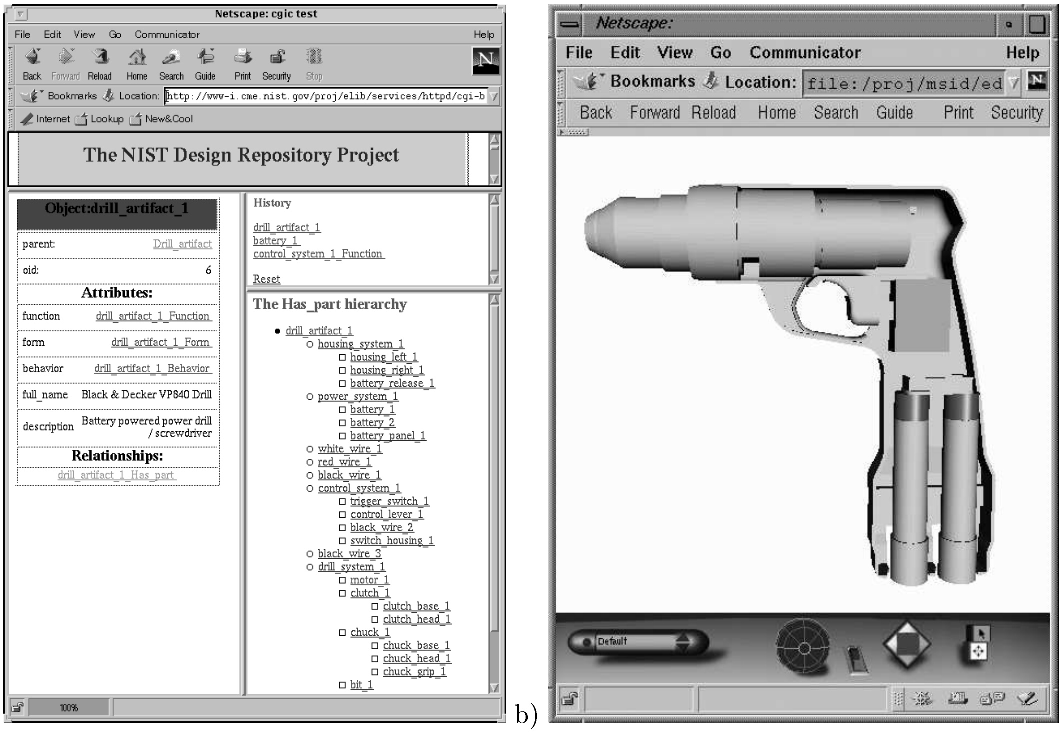
a)  b)

Figure 8: Web-based Design Repository Browser a) multiframe window b) visualization of the *drill_artifact_1* form using VRML

entity. The browser will then retrieve the information for that new entity and will update the browser view with the new entity. The middle right frame contains the history of the user's browsing session, allowing the user to return to any previously viewed entity by clicking on an entity name. The bottom right frame includes a view of the appropriate hierarchy allowing to locate the selected object within the design structure. In this instance, the user is viewing an artifact object, so the `Has_part` artifact hierarchy is shown. In this frame, too, the designer can jump to a different part of the artifact representation by clicking on an entity name. Within all three of the dynamic frames, hyperlinks to different types of entities are color-coded, which makes easier to distinguish them from each other.

4.4 Implementation Example

The prototype design repository that has been used in the examples in this paper represents the Black & Decker VP840 cordless power drill. This repository was created using the Design Repository Editor and contains 28 artifact objects (assemblies, subassemblies and components) and over 250 additional entities (objects and relationships) belonging to 64 classes and 6 relationship classes. Most of the objects were not created explicitly by the designer, but were initially generated automatically as "dummy" objects and then completed with detailed descriptions by the user. Similarly all relationships were created automatically by the editor as described previously. The class, function and artifact hierarchies for the power drill are depicted in Figs. 7a,7b, and 8a respectively. Fig. 8b shows the visualization of the geometry of the modeled product, stored in VRML format.

5 Summary and Future Research

The practical use of design repositories in industry requires an information modeling framework (including a modeling language to formalize engineering knowledge and model engineering artifacts), and a computational platform to create, store, and retrieve this knowledge. In the framework of the NIST Design Repository Project the Data Language and the Design Representation Language have been developed to defining data structures describe the engineering context and model artifacts in a design domain. This allows to represent structures of artifacts, functions, flows, relationships and other entities described in previous sections in the formatted text. Initial work has been done in developing a comprehensive taxonomy of functions and associated flows. The existing taxonomies can be extended with new terms or new domains using the data structures that have been established. Finally web-based interfaces were implemented to facilitate the creation and and retrieval capture of design information by distributed users via the Internet. To illustrate the functionality of the modeling framework an example repository was created.

The basic research in the area of design repositories that has been conducted in the framework of the NIST Design Repository Project focuses on development of knowledge representations and a computational platform for data handling via the World Wide Web. The motivation for developing design repositories (rather than traditional design databases), was discussed in Section 1 of this paper. However, the current implementation, described in section 4, still makes use of traditional database technology for information storage. Currently, this limits the locations of design repositories to single sources where database servers are available. Future emphasis will be placed on taking greater advantage of the distributed nature of the World Wide Web by distributing design repositories to a greater extent. To improve the information circulation and to distribute sources of knowledge the development of a new version of the system architecture has recently begun.

The new system architecture will differ from the current one in two significant ways. Both of these changes are strongly motivated by the needs of the engineering industry (as will be described below). The first is a migration from the current repository text file format to an Extensible Markup Language [29] (XML)-based text file. The motivation for this is that the schemata used to represent information in the current format are not readily used other than by applications developed as part of this project by people with knowledge about the format. In contrast, XML has the advantage of widespread adoption in the information technology world and appears to be on its way to becoming a World Wide Web Consortium (W3C) standard. Third-party XML parsers, compilers, and authoring tools exist and/or are under development, and built-in XML support is expected in upcoming versions of several commercial web browsers and word processing applications. This change will result in a more broad-based solution for the engineering industry, in which companies are increasingly looking towards purchase of off-the-shelf software over in-house development when possible.

The second change is the decoupling of the database system from the Design Repository Project editing and browsing tools. The XML-based artifact text file will be able to act as a self-contained database, allowing browsing and editing of artifact repositories without requiring a commercial database management system (DBMS). One advantage is that for small-scale applications, a company will not have to invest in a DBMS to take advantage of the design repositories. For large scale applications, DBMSs are quite useful for management of large amounts of data, providing capabilities such as transaction management, consistency maintenance, distributed databases, synchronization of mirrored databases, information caching, and more. In these cases, the decoupling is still desirable because use of the Design Repository Project tools will not tie a company to a specific commercial DBMS. The Design Repository Project tools provide the interface between the user and XML-based artifact text file. Then, for a given commercial DBMS, a compiler would be developed to exchange information between the file and a true database in order to take advantage of the features of commercial database systems.

Another important possibility falls out from this new architecture: the ability for companies that are using design repository technologies to share and exchange artifact data even if they use different DBMS systems in-house, by using the XML-based artifact text file as a neutral file format for information transfer. As industry relies more and more on partnerships and outsourcing with other companies, interoperability across corporate boundaries and the ability to exchange design information – not just geometry, but knowledge

– becomes increasingly important. This solution will help provide this interoperability without requiring partners to adopt the same DBMSs in-house.

Another objective for future work is to incorporate a more formal representation of behavior to allow composable simulations. Work in this area is being done as part of two projects that are funded by the Defense Advanced Research Projects Agency (DARPA) Rapid Design Exploration and Optimization (RaDEO) program, for which NIST is a funding agent: the Model-Based Support of Distributed Collaborative Design (previously How Things Work) project at the Stanford University Knowledge Systems Laboratory [13], [5], and the Active Catalog project at the University of Southern California Information Sciences Institute [20],[6]. The behavior representation languages developed in these projects are among the options being considered for representing behavior within the NIST Design Repository Project.

References

[1] Alberts L.K. and F. Dikker (1992), "Integrating Standards and Synthesis Knowledge Using the YMIR Ontology," *Artificial Intelligence in Design '92*, J.S. Gero (ed.), Kluwer Academic Publishers, Boston.

[2] 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.

[3] Bliznakov, P. I., J. J. Shah, and S. D. Urban (1996), "Integration Infrastructure to Support Concurrence and Collaboration in Engineering Design," *Proceedings of the 1996 ASME Design Engineering Technical Conferences and Computers in Engineering Conference*, Paper No. 96-DETC/EIM-1420, Irvine, CA, August.

[4] Chandrasekaran, B., A. Goel, and Y. Iwasaki (1993), "Functional Representation as Design Rationale" *IEEE Computer*, January, pp. 48-56.

[5] Chouiery, B. Y., S. McIlraith, Y. Iwasaki et al. (1998), "Thoughts on a Practical Theory of Reformulation for Reasoning about Physical Systems," *Proceedings of the Symposium on Abstraction, Reformulation, and Approximation (SARA '98), pp. 25-36; revised version to appear in Proceedings of the 12th International Workshop on Qualitative Reasoning (QR '98).*

[6] Coutinhi, M., R. Eleish, J. Kim, V. Kumar, S. R. Ling, R. Neches and P. Will (1997), "Active Catalogs: Integrated Support for Component Engineering," *Proceedings of the 1998 ASME Design Engineering Technical Conferences*, Paper No. DETC98/CIE-5521, Sacramento, CA, September.

[7] 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.

[8] Goel, A., S. Bhatta, and E. Stroulia (1996), "KRITIK: An Early Case-Based Design System," To appear in *Issues in Case-Based Design*, M. Maher and P. Pu (eds.), Hillsdale, NJ, Erlbaum.

[9] 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.

[10] Gorti S. R., G. J. Gupta, G. J. Kim, R. D. Sriram, and A. Wong (1998), "An Object-Oriented Representation for Product and Design Process," to appear in *Computer Aided Design*.

[11] 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.

[12] 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.

[13] Iwasaki, Y., A. Farquhar, R. Fikes, and J. Rice (1997), "A Web-Based Compositional Modeling System for Sharing of Physical Knowledge," *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, AAAI Press, pp. 494-500.

[14] ISO 10303, *Industrial Automation Systems and Integration - Product Data Representation and Exchange*, ISO TC184, SC4 (ISO Technical Committee 184, Subcommittee 4).

[15] ISO 10303-203:1994, Industrial Automation Systems and Integration - Product Data Representation and Exchange - Part 203: Application Protocol: Configuration Controlled Design, ISO TC184, SC4 (ISO Technical Committee 184, Subcommittee 4).

[16] ISO/IEC 14772, *Information technology - Computer graphics and Image Processing - The Virtual Reality Modeling Language*, ISO/IEC JTC1, SC24 (ISO/IEC, Joint Technical Committee 1, Subcommittee 24).

[17] 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.

[18] Kirschman C. F. and G. M. Fadel (1998), "Classifying Functions for Mechanical Design," *ASME Journal of Mechanical Design*, 120(3):475-482.

[19] Little A., K. L. Wood, and D. McAdams (1997), "Functional Analysis: A Fundamental Empirical Study for Reverse Engineering, Benchmarking and Redesign," *Proceedings of the 9th ASME International Conference on Design Theory and Methodology*, Sacramento, CA, September.

[20] Ling, S. R, J. Kim, P. Will, and P. Luo (1997), "Active Catalog: Searching and Using Catalog Information in Internet-Based Design," *Proceedings of the 1997 ASME Design Engineering Technical Conferences*, Paper No. DETC97/CIE-4292, Sacramento, California, September.

[21] Murdock J. W., S. Szykman, and R. D. Sriram (1997), "An Information Modeling Framework to Support Design Databases and Repositories," *Proceedings of the 1997 ASME Design for Manufacturing Conference*, Sacramento, CA, September.

[22] Pahl G. and W. Beitz (1988) *Engineering Design: A Systematic Approach*, Springer Verlag, New York.

[23] 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*, 28(5).

[24] Stone R. B., K. L. Wood, and R. H. Crawford (1998), "A Heuristic Method to Identify Modules from a Functional Description of a Product," *Proceedings of the 10th ASME International Conference on Design Theory and Methodology, Atlanta*, GA, September.

[25] 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.

[26] Szykman, S., R. D. Sriram, C. Bochenek and J. W. Racz (1998), "The NIST Design Repository Project," *Advances in Soft Computing - Engineering Design and Manufacturing*, Springer-Verlag, London (in press).

[27] Wong A. and R. D. Sriram (1993), "SHARED: An Information Model for Cooperative Product Development," *Research in Engineering Design*, 5(1) pp.21-39.

[28] Wood III, W. H. and A. M. Agogino (1996), "Case-Based Conceptual Design Information Server for Concurrent Engineering," *Computer-Aided Design*, 28(5).

[29] World Wide Web Consortium (1998), *Extensible Markup Language (XML) 1.0*, World Wide Web Consortium (W3C) Recommendation, February.