

DETC2006/CIE-99518

Introduction of a Data Schema: The Inner Workings of a Design Repository

Matt R. Bohm,
and **Robert B. Stone, Ph.D.**
Design Engineering Laboratory
Department of Interdisciplinary Engineering
University of Missouri – Rolla
Rolla, MO 65409

Timothy W. Simpson, Ph.D.
and **Elizabeth D. Steva**
Engineering Design Optimization Group
Department of Mechanical & Nuclear Engineering
The Pennsylvania State University
University Park, PA 16802

ABSTRACT

This paper describes the fundamental pieces of design information that compose the University of Missouri Rolla's (UMR) design repository schema. Knowledge-based systems along with similar design information-capture schemas are reviewed. Repository schema conventions and specific implementation details are outlined to provide an understanding of the system connections and information relationships. Next, the repository schema is divided into seven main categories of design information including: artifact-, function-, failure-, physical-, performance-, sensory- and media-related information types. Each of the seven types of design information are described in detail to illustrate what elements of design information are recorded and how their relationships are established. An overview of the entire repository database is also presented. The result is a complete description and specification of the repository framework and allowable design information types. Finally, a brief comparison is made between the UMR repository and its antecedent NIST repository framework.

1 INTRODUCTION

The collection of artifact design data has been a continuing process at UMR, first starting in 1999 [1]. Since that time, the process in which artifact data is gathered and recorded has changed significantly. Artifact design data was originally recorded in Excel spreadsheets and mainly took the form of Bills of Materials (BOM), Function Component Matrices (FCM), and Design Structure Matrices (DSM). A design repository initiative by the National Institute of Standards and Technology (NIST) helped to guide the design repository project at UMR to a more mature state. Because of the NIST initiative, UMR implemented a front-end database entry approach capable of generating XML design data in accordance with NIST's defined format. With the front-end database entry method in place, the UMR design repository project then moved towards a back-end database for design knowledge storage, a front-end entry application for design knowledge entry, and a web front-end for design knowledge retrieval and artifact browsing.

Each transformation of the repository project has allowed for more types of design knowledge and data relationships to be recorded.

Recently the repository project has become a collaborative effort between UMR, UT-Austin [2, 3], Penn State, Virginia Tech and Bucknell [4]. Currently, the design repository serves as a hub for designers for knowledge exchange and design tool generation. Knowledge exchange and design tool generation occur in two main areas. Design knowledge and information must first be generated and then presented in a form that is easy for designers to navigate and use. Information entry occurs within a front-end entry application while information retrieval occurs over the Internet through the design repository's web portal (<http://function.basiceeng.umn.edu/repository>). The hub of these two applications is the design repository database and more specifically the database schema. The database schema establishes what types of design information can be stored and how those elements relate to one another.

The goal in this paper is to present and fully describe the repository powering database schema version 2.0¹. The newest database schema represents underlying design information types, relationships, and structures. Because the schema represents the fundamental underpinnings of the repository, it is necessary to report and describe the schema's composition and implementation. This paper reports on research efforts to 1) identify pieces of fundamental design information that support designer activities, 2) segment and classify the pieces of design information, 3) define relationships between the disparate pieces of design information, 4) develop ways to standardize design information representation, and 5) deliver a functional database schema. Sections 3 and 5 present a broad overview of the repository, Section 4 provides implementation level details and specifics and Section 6 compares the UMR repository to the original NIST repository framework.

2 BACKGROUND

In this section we review three different types of knowledge-based systems: 1) domain specific knowledge based systems, 2) Product Data Management (PDM) systems, and 3) a similar design repository system at NIST. All of the systems described are different in implementation and purpose, but all exist to store elements of design knowledge for future use and collaboration. The intent of the UMR

¹ The original repository (a collection of files from various applications) was more of a specification than a schema. Version-wise it was 0.X. The original database schema reported in [5], represents the release of version 1.x of the schema. Currently the repository is migrating to version 2.0 of the schema.

design repository, and more specifically, the data schema is the same as the reviewed systems, namely, to enable the storage of design information for reuse and collaboration.

2.1 Knowledge storing systems

Several researchers have built a variety of knowledge-based design information systems and have used different product representations with varying degrees of design knowledge abstraction. Although not all of the knowledge bases built were designed for the collection of function-based design, information can still be extracted from these systems and their representations.

Summers [6] reports on a feature-based knowledge system designed for CAD-based feature elements with the intent of supporting designer activities. As conceptual designers, features are crucial pieces of modeling information. The modeling approach between a CAD-based feature system and a functional-based representation are different; however, both capture relevant design information. Design knowledge in both cases is highly relevant to the respective fields of study. What differentiates the two approaches is the application of the design representations. The CAD-based knowledge system from Summers exemplifies that all design knowledge is relevant dependent upon the domain, representation, and level of abstraction. Dixon [7] makes the point that feature descriptions and representations must be valid within their respective domain of use.

Architects, like designers, also have the need to store valuable design knowledge. Cherneff [8] describes an architectural knowledge-based system, and—even though this system may seem very distant from function based knowledge systems—the underpinning philosophy in both is to store design information for reuse and collaboration. Cherneff [8] explains that design abstractions all require context and that objects only make sense in the object schema in which they are defined. Again, he is echoing the need to develop design representations that are relevant to specific domains.

Functional representations have been used to represent design knowledge in early repository systems. These systems used a block diagram approach and were based on “function logic.” One of these early systems, described by Sturges [9] and powered by Hypercard stacks, was used to navigate function diagrams. In this preamble to the Functional Basis and defined functional modeling techniques, a representation schema had to be chosen. The representation schema used by Sturges [9] built on function logic to describe complex systems and included mathematical relationship equations in relationship to the “function blocks.” Through the use of “function logic” and “function blocks” designers were able to gain insight on how a product operates functionally.

Regli [10] in partnership with NIST and the National Science Foundation (NSF) has also developed a CAD-based design repository. The focus of Regli’s work includes collaboration in the field of CAD, engineering design, manufacturing process planning, and feature recognition. The design repository contains mostly CAD, solid models, and assemblies along with some supporting documentation such as cost and assembly plans.

In all of these cases, a knowledge system was designed and built to provide domain specific information to designers. The knowledge systems presented vary not only in domain but also complexity [11]. The breadth and abstraction of the knowledge systems vary, but all rely on domain specific information for use in domain specific applications. Candy [12], conducts a thorough study of strategic knowledge representations. According to Candy, there are certain criteria that a representation should include regardless of whether the representation is “of” or “for” design. The three point test for a design representation described by Candy states that: 1) a representation should be accurate and include the essential features and or attributes of system, 2) a representation should not confuse and aim for clarity while cautious of over simplification, and 3) the style of the representation should be appropriate for the intended purpose and context of use.

2.2 Product Data Management Systems

In recent years product data management (PDM) systems have emerged to help store and retrieve product and part data. PDM systems allow for part hierarchy storage as well as process data and project management elements. Svensson and Malmqvist [13] explore a PDM system and demonstrate many uses of such a system. The PDM system demonstrated collects requirement, function, concept, and part structures as well as property models. Additionally a PDM system stores the entire product structure, variants, revisions, and finally documentation and CAD models. Although function structures and property models can be stored within a PDM system, they are not capable of storing the detailed function based information we desire and integrating it into useful design tools without heavy modification. A PDM system is a highly effective tool for use in the manufacturing side of emerging products and parts but is fundamentally different from a repository system. Within the UMR repository, we store similar pieces of design knowledge like CAD models and develop part hierarchies; however, our main focus is the mapping between functions and components and the compatibility of components to connect together as a system.

2.3 Similar Design Repository Definition

The most similar system to UMR’s repository schema takes form in the NIST design repository representation model [14, 19]. The NIST design repository representation model is a basic framework to help guide what type of product information is collected and how the elements of information are related to each other. NIST has also developed a mapping from this representational framework into an XML (eXtensible Markup Language) data format. Within the NIST XML code there are five different sections: Artifacts, Functions, Forms, Behaviors and Flows. These sections contain information relative to their denoted naming system and are visited in more detail in Section 6.

3 UMR DESIGN REPOSITORY CONVENTIONS

The UMR design repository project is an artifact-centric repository, meaning that for a design attribute to exist, it must be linked to an artifact. Other data types contained in the design repository, such as manufacturing and physical parameters for example, describe additional design attributes while still relating to their artifact hub. Because the design repository is artifact-centric, understanding the artifact table and associated relationships is key to understanding the data handling capabilities of the repository.

Both the current and emerging versions of the repository are built and served by a PostgreSQL (a SQL variant) database [20]. In general, the database contains tables that have clusters of similar types of information. Database tables are then connected to other tables within the database to form data relationships. In this paper there are two types of database tables described: 1) A database table description, which refers to the fields and associated data types that define a database table (alternatively, for non-computer scientists, table descriptions describe the structure and connectedness of the data), and 2) a database table that refers to the set of data entries in the database (alternatively, the actual data that is entered into the repository – in this case, the product data). The term row will be used to refer to an entry in a database table and the term field will be used to refer to an object in the database table description. Table 1 shows the artifact table description in the repository database schema (version 2.0).

All table descriptions throughout this paper will be presented in the same format as Table 1. The first column represents the field name, the second column specifies the data type, the third column denotes whether or not a piece of information is mandatory, and the fourth column describes any default values, if applicable. The fifth column describes the type of key that might exist for a particular field. Only one primary key can exist per table and is used to develop a unique reference to the particular entry in that data table. Having a foreign key designation means that the particular field references the

Table 1 – Artifact table description

artifact				
field name	data type	mandatory	default value	key type
id	serial	yes	N/A	primary
name	vchar	yes	N/A	N/A
child_of_artifact	int	no	N/A	foreign
basis_name	int	no	N/A	foreign
serial_id_number	vchar	no	N/A	N/A
assembly	boolean	yes	FALSE	N/A
description	vchar	no	N/A	N/A
quantity	int	yes	1	N/A
system	int	yes	N/A	foreign
manufacturer	vchar	no	N/A	N/A
trademark	vchar	no	N/A	N/A
artifact_release_date	date	no	N/A	N/A
entry_date	date	yes	N/A	N/A
modification_date	date	no	N/A	N/A
creator_info	int	yes	N/A	foreign

primary key of another data table. All data tables in the UMR design repository begin with an id column that is a serial integer, mandatory for any information entry with no default value and designated as the table's primary key.

In the UMR design repository implementation, there are two main categories of tables—those that store artifact-specific design data information and those that store taxonomies and bases that classify design information. The tables that store taxonomies and bases are denoted with *_type* after the table name. Figure 1 shows all 41 of the repository data tables with the 13 data storing tables highlighted. Taxonomy and basis storing tables do not reference design data storing tables; however, they may reference themselves in order to establish hierarchies. Shown in Table 2 is a prime example of a basis-storing table: the *subfunction_type* table description.

Table 2 – Subfunction type table description

subfunction_type				
field name	data type	mandatory	default value	key type
id	serial	yes	N/A	primary
subfunction	vchar	yes	N/A	foreign
tier	int	yes	N/A	N/A
child_of_subfunction	int	no	N/A	foreign
definition	vchar	no	N/A	N/A

As with all other tables in the repository, the *subfunction_type* table begins with a serial id that establishes a primary key. The second field of the table is where the actual Functional Basis term is stored. Tier is used to denote whether the particular Functional Basis term is in the primary, secondary, or tertiary level [21]. Child of subfunction establishes a hierarchy of the Functional Basis terms and the definition field is used to hold the definition of the particular function. Table 3 offers another view of the *subfunction_type* table containing a snippet of the actual functional basis implemented in the repository.

Table 3 differs from Table 2 in that Table 2 is a table description where Table 3 is a populated database table. Starting with the first row of id 1, the subfunction name is 'branch' and is at the top-level tier of the functional basis. Branch is not a child of any other subfunction, which is denoted by using 'null' in the *child_of_subfunction* column. Note, that in order to reduce the size of the table, the actual subfunction definitions have been removed, however the placeholder is still shown. Row ids 2 through 9 are also denoted as primary level functions in accordance with the Functional Basis. Beginning at row id 10, 'separate' is shown of being in the second tier (secondary level) of the Functional Basis and having row id 1 as a parent. This convention follows the Functional Basis in that 'separate' is a secondary function under 'branch.'

4 UMR DESIGN REPOSITORY DATA GROUPS

The design information captured by the design repository data schema can be classified into seven main groups: artifact-, function-, failure-, physical-, performance-, sensory- and media-related information types. All seven of these categories are represented in different data tables but are all brought together by the use of data table relationships, found in the table descriptions for each table. In this sec-

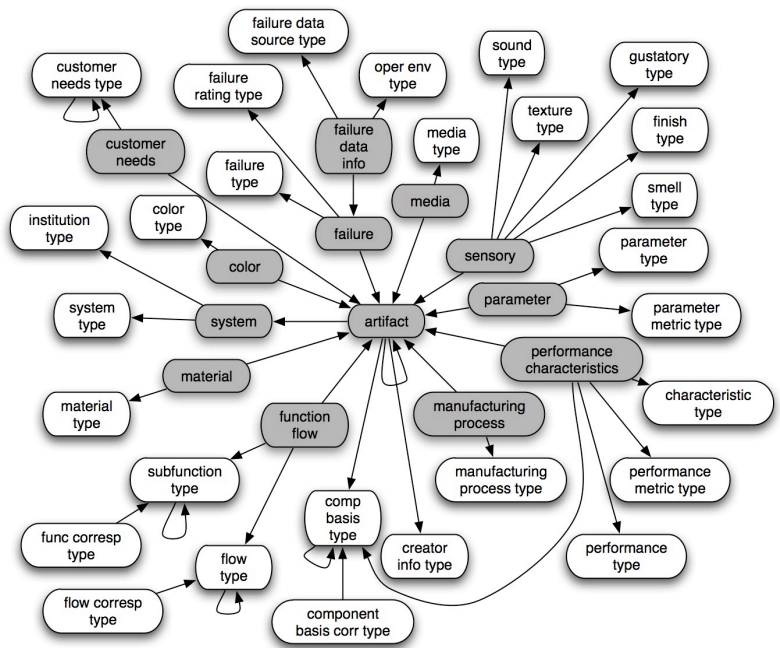


Table 3 – Subfunction type database table snippet

tion, each of the seven data type categories are reviewed along with the specific pieces of information they hold. Section 5 details how these elements are connected together to create a cohesive design repository.

4.1 Artifact-related design knowledge

As mentioned in Section 3, the artifact table serves as a central hub for the remaining six categories of data. Although all design information typically references an artifact, there are a few pieces of design information that the artifact table stores directly. Information is recorded on an artifact-by-artifact basis in the artifact table. An artifact can be considered an entire product, a sub-assembly, or a single part when stored in the design repository. To represent the artifacts in a product in the repository, the product is first identified as an artifact, and then all individual assemblies, sub-assemblies, and artifacts are grouped accordingly under that artifact. The repository database has the capability of establishing parent-child relationship such that a product artifact hierarchy is created. In order to keep a strict separation of different products within the repository a system table is used, the system table description is shown in Table 4. The separation of products allows for private systems such as those modeled for specific industry partners to be kept hidden from public view and also allows for segmentation of system categories.

Table 4 – System table description

system				
field name	data type	mandatory	default value	key type
id	serial	yes	N/A	primary
name	vchar	yes	N/A	N/A
system_type	int	yes	N/A	foreign
description	vchar	no	N/A	N/A
contributing_institution	vchar	yes	N/A	N/A

Looking back at the artifact table description shown in Section 3 (see Table 1), there is a placeholder for a system reference for each artifact instance. For each new product that is added to the repository a unique system id is established. Every artifact belonging to the given system is then referenced to the system id. In the system table description (see Table 4), a system name, system description, and contributing institution associated with the system. For example there may be 30 artifacts named 'motor,' all of which have a unique artifact id and belong to different products within the repository. The *contributing_institution* field in the system database table is used to track

Table 3 – Subfunction type database table

subfunction_type				
id	subfunction	tier	child_of_subfunction	definition
1	branch	1	null	subfunction definition
2	channel	1	null	subfunction definition
3	connect	1	null	subfunction definition
4	control_magnitude	1	null	subfunction definition
5	convert	1	null	subfunction definition
6	provision	1	null	subfunction definition
7	signal	1	null	subfunction definition
9	support	1	null	subfunction definition
10	separate	2	1	subfunction definition
11	distribute	2	1	subfunction definition
12	import	2	2	subfunction definition
13	export	2	2	subfunction definition
14	transfer	2	2	subfunction definition
15	guide	2	2	subfunction definition
16	couple	2	3	subfunction definition

what institutions have recorded design information for a particular product. The system database table also includes a system_type field. The system_type field links to the system_type database table containing a list of different product categories. Example product categories include consumer, industrial, automotive, etc.

The artifact table description (see Table 1) begins with a serial based id number to establish unique serial number for each artifact that resides in the database. Moving through the artifact table, data attributes such as the artifact name, description, quantity, manufacturer, trademark, artifact release date, entry date and modification date are present. All of these data types are straightforward and are identifiable by their field names and contain no external table references.

The child_of_artifact database table is used to create an artifact hierarchy; this is accomplished by designating the field as a foreign key, which in this instance points to an artifact id. For example, a generic housing may be represented with an artifact id of 1000 and a component within that housing has an artifact id of 1001. The child_of_artifact field for artifact id 1001 would point to artifact id 1000 to establish a parent relationship.

Next in the artifact database table, the basis_name field is used to associate a generic artifact name to a specific artifact [22]. For example an artifact denoted as a coffee cup would reference the component_basis_type table to establish that ‘reservoir’ is the corresponding component basis term. The serial_id_number field is used to support previous design repository information that was not originally encoded with a primary key. When an artifact is a grouping of several artifacts, the assembly field is used. The assembly field Boolean value defaults to FALSE indicating that the artifact is a singular artifact. For book-keeping purposes, the creator_info field is used. The creator_info field references the creator_info_type table, which contains contributor information such as their name, email address, and affiliation.

4.2 Function-related design knowledge

Product functionality is very important not only to conceptual design but also to other design and optimization methods that use function as a link to existing design knowledge. Some of these methods include the Function Failure Design Method (FFDM) [23, 24], the Risk in Early Design (RED) [25] tool and modern concept generation techniques [2, 3]. Since several aspects of design engineering and product design revolve around function, it is highly necessary to accurately represent artifact functionality digitally.

The function_flow data table in the repository is used to allow portions of functional models to be associated with an artifact. In order to accurately capture the material, energy, and signal flow through a product, it is necessary to have additional artifact connection information alongside the standard function and flow language. Capturing the function, flow, and artifact connection information is done by associating an input and output artifact and flow with each function. Table 5 shows the function_flow table description.

Table 5 – Function flow table description

function_flow				
field name	data type	mandatory	default value	key type
id	serial	yes	N/A	primary
describes_artifact	int	yes	N/A	foreign
supporting	boolean	yes	FALSE	N/A
input_artifact	int	yes	N/A	foreign
input_flow	int	yes	N/A	foreign
subfunction	int	yes	N/A	foreign
output_flow	int	yes	N/A	foreign
output_artifact	int	yes	N/A	foreign

Similar to the artifact table, the function_flow table begins with a serial id number creating a unique primary key. The primary key ensures that each set of function and flow descriptions are represented uniquely in the scope of the entire set of function-flow descriptions in the repository. Each tuple containing the {input_artifact, input_flow, subfunction, output_flow, and output_artifact} is linked to a specific artifact by the describes_artifact field. The supporting field is used to establish whether a particular function tuple is described as a supporting or conceptual function. The supporting field also has a default value of FALSE, which corresponds to a function being recorded as a conceptual function. The input_artifact and output_artifact fields are both foreign keys that reference a specific artifact id number in the artifact table. Input_flow and output_flow are also foreign keys but reference a specific flow id in the flow_type table. The subfunction field is also a foreign key and references a specific function id in the function_type table. All of the data elements in the function_flow table are specified as mandatory in order to accurately represent functionality. In cases where an artifact solves multiple functions the input and output artifact fields can be designated as ‘internal,’ representing that a particular flow stays within an artifact’s boundary. If an input (or output) flow comes from (or goes to) more than a single artifact, the flow can be designated as going to (or from) multiple sources using the ‘internal’ designation. For example, when an artifact has an incoming flow of electrical energy from two specific sources both electrical energy flows would be transferred to the designated artifact with a destination of ‘internal.’ Functionality of the artifact can then be recorded using ‘internal’ as the input flow. When multiple artifacts are used in concert to solve a single function each artifact is denoted with the overall function.

Table 6 shows the function_flow table populated with sample data to demonstrate how function relationships are generated. Reading across the table, the two filled in sample function and flow tuples describe artifact number 0000008. In the row beginning with an id of 1, the input_artifact corresponds to ‘external’ and the output_artifact corresponds to 0000009. If an input or output artifact is denoted as ‘external’ it means that a particular source or destination of a flow crosses the given product’s boundary. Both input_flow and output_flow reference id numbers in the flow_type table. For this example, flow id of 16 corresponds to ‘electrical energy.’ The subfunction field in Table 6 references an id number in the subfunction_type table, with an id of 12 representing the function ‘import.’ All of these designations for row id 1 correspond to ‘electrical energy’ being imported from an outside source with a destination of an artifact having an id of 0000009.

Moving on to row id 2 of Table 6, the artifact being described has an id number of 0000009, an input artifact id of 0000008, input flow of id 16, subfunction id of 22, output flow id of 44, and a destination artifact id of 0000006. Translating the id numbers, the row reads as having an input flow of ‘electrical energy,’ the subfunction ‘convert’ and an output flow of ‘rotational mechanical energy.’ Adding both of these rows together shows that two separate artifacts are being described: one that would take form as an electric plug or wire (artifact id 0000008) and the other artifact taking form as some kind of electric motor (artifact id 0000009). The input artifact for the electric cord is external while the output artifact is the motor. The electric motor has a source artifact of the electric cord while the destination artifact, specified as artifact id 0000006, would likely be some kind of coupler, gear, or other artifact that can connect to an electric motor. A translated ver-

Table 6 – Function flow database table with sample data

function flow							
<i>id</i>	<i>describes</i>	<i>input_artifact</i>	<i>input_flow</i>	<i>subfunction</i>	<i>output_flow</i>	<i>output_artifact</i>	<i>supporting</i>
1	0000008	external	16	12	16	0000009	FALSE
2	0000009	0000008	16	22	44	0000006	FALSE
3	xx	xx	xx	xx	xx	xx	xx
4	xx	xx	xx	xx	xx	xx	xx
5	xx	xx	xx	xx	xx	xx	xx

Table 7 – Function flow database table with translated sample data

function flow							
<i>id</i>	<i>describes</i>	<i>input_artifact</i>	<i>input_flow</i>	<i>subfunction</i>	<i>output_flow</i>	<i>output_artifact</i>	<i>supporting</i>
1	electric cord	external	electrical energy	import	electrical energy	electric motor	FALSE
2	electric motor	electric cord	electrical energy	convert	rotational mechanical energy	0000006	FALSE
3	xx	xx	xx	xx	xx	xx	xx
4	xx	xx	xx	xx	xx	xx	xx
5	xx	xx	xx	xx	xx	xx	xx

sion of Table 6 is shown in Table 7. For both Tables 6 and 7, the functions are described as conceptual functions, taking the value of FALSE in the supporting field [26].

4.3 Failure-related design knowledge

Failure information in this design repository is driven by efforts including the FFDM [23, 24], RED [25], and adaptations of modern Failure Modes and Effects Analysis (FMEA) [27] techniques. It is necessary to build an infrastructure such that designers and engineers can archive and easily access this critical information. A failure mode taxonomy for mechanical and electrical components has been developed at UMR and is used as the reference taxonomy in this work [23, 24].

Like the function_flow table, the failure database table description shown in Table 8, begins with a serial id number and link to the particular artifact being described (describes_artifact). It is necessary that the serial id and describes_artifact fields are present to establish a unique identifier for a given set of failure information and to properly link the failure information to a specific artifact. Next, the particular type of failure is recorded in the failure field. Again, the failure field actually references the failure taxonomy, meaning that only the failure id number is actually entered in failure table.

The next two fields in the failure database table are used to specify the severity and whether the failure mode is an actual or potential failure mode. Typically a 1-5 scale is used to denote severity; however, the failure data table allows a float value to be entered in the severity field. The float value is allowed because not all data contributions are rated on the same 1-5 severity scale. It is necessary to specify whether a particular failure mode is an actual failure mode or is only noted that it ‘could’ happen. Because of this very distinct difference of actual versus potential implication it is necessary to record the correct information. The potential field in the failure data table is a Boolean and has a default value of FALSE indicating that a failure mode is an actual failure mode.

In cases where a failure mode is denoted as an actual failure mode, it is necessary to record the number of occurrences, the sample size, and rating type. The occurrences field in the failure data table is used to record the integer number of occurrences of a particular failure mode that an artifact has demonstrated. In addition to recording the number of occurrences, recording the sample size is highly relevant and more useful to the designer. Sample size is recorded in the sample_size field and takes an integer value.

The default rating scale assumed in the repository is the 1-5 severity scale [27]. In cases where an alternate failure rating scale is used, the rating_type field in the failure table can be used to reference the rating_type table. The rating_type table can be populated with a list of failure mode rating types, a description of the rating system, and conversion values to the repository standard 1-5 rating scale. When occurrence and sample size data is not available and only failure rate data is specified, the rate field is used to input a float value of the fail-

ure rate. Ideally it is better to have occurrence and sample size failure data such that similar artifacts and functions can be clustered to present statistically valid failure likelihood and severity information.

Table 8 – Failure table description

failure				
<i>field name</i>	<i>data type</i>	<i>mandatory</i>	<i>default value</i>	<i>key type</i>
id	serial	yes	N/A	primary
describes_artifact	int	yes	N/A	foreign
failure	int	yes	N/A	foreign
severity	float	no	N/A	N/A
potential	boolean	yes	FALSE	N/A
occurrences	int	no	N/A	N/A
rating_type	int	yes	1	foreign
sample_size	int	no	N/A	N/A
rate	float	no	N/A	N/A

When more accurate failure mode information is available, the failure_data_info database table, shown in Table 9, can be used to record the additional information. The failure_data_info table is used to supplement information in the failure data table by adding the data source, a report number for bookkeeping, the operational environment of the artifact at time of failure, date of incident, and a description. Unlike the failure data table, which references artifact id numbers, the failure_data_info table references a particular failure id number. Using this referencing scheme means additional failure_data_info information can only be associated with existing failure mode documentation in the failure data table.

The driving force behind the failure_data_info comes from NASA, industry partners, and other academic institutions [2830]. For critical subsystems it is necessary to accurately record not only the failure modes but also additional descriptions of the failure mode. The data_source field in the failure_data_info table references a data_source_type table. Data source types may take on the form of corporate-specific failure databases, warranty data, or NASA’s Problem Failure Reporting (PFR) database [31]. To be accurate in recording specific failure events in the repository, a report_number can be included to further document the failure mode. The report_number would include institution specific labeling schemes for their particular data sources of a documented failure incident. The operational_environment is included in the failure_data_info data table to further denote how or where an artifact or product was being used at the time a failure event occurred. The oper_env field references the oper_env_type table, a list of various operational environments. No taxonomy exists for the operational environment type at this time but as new items are added to the repository that include an operational environment the oper_env_type table is updated. Further research efforts may include developing an operational environment type taxonomy. The date_of_incident and description fields in the failure_data_info table are used to record the date of a particular failure event along with any free text description of the event.

Table 9 – Failure database info table description

failure_data_info				
field name	data type	mandatory	default value	key type
id	serial	yes	N/A	primary
describes_failure	int	yes	N/A	foreign
data_source	int	yes	N/A	foreign
report_number	varchar	no	N/A	N/A
oper_env	int	no	N/A	foreign
date_of_incident	date	no	N/A	N/A
description	varchar	no	N/A	N/A

4.4 Physical-related design knowledge

There are several types of physical-related design knowledge elements that can be used to describe artifacts. Currently the repository records four main categories of physical design information elements: manufacturing information, artifact material, rough geometric bounding dimensions, and color.

The manufacturing_process table description, shown in Table 10, is used to denote specific manufacturing processes utilized in the manufacture of the referenced artifact. Similar to most other database tables in the repository, the manufacturing_process table begins with a serial id number and a reference to a specific artifact id (describes_artifact). The manufac_process_type field in the manufacturing_process table references an id of a specific type of manufacturing in the manufacturing_type table. Examples of manufacturing types include casting, machining, injection molding, etc. The manufacturing_process table is used only to link a specific artifact to a type of manufacturing process; process data types are stored only in the manufacturing_type table. It is not required to specify a manufacturing type when recording artifact information; however the repository can record multiple manufacturing processes for each artifact.

Table 10 – Manufacturing process table description

manufacturing_process				
field name	data type	mandatory	default value	key type
id	serial	yes	N/A	primary
describes_artifact	int	yes	N/A	foreign
manufac_process_type	int	yes	N/A	foreign

The material database table description, shown in Table 11, operates in the same manner as the manufacturing_process database table but is used to link an artifact to material types instead of manufacturing types. Examples of material types in the material_type table include ABS plastic, aluminum, stainless steel, etc. Like the manufacturing_process table, it is not required to specify material information when recording artifact data although multiple material types can be specified for each artifact. For example, when an artifact is an assembly that contains multiple parts, it may be advantageous to note that the assembly consists of ABS plastic and polycarbonate plastic artifacts.

Table 11 – Material table description

material				
field name	data type	mandatory	default value	key type
id	serial	yes	N/A	primary
describes_artifact	int	yes	N/A	foreign
material	int	yes	N/A	foreign

The color table is similar to the manufacturing_process and material database tables and is shown in Table 12. Each instance of a color association is tracked by the serial id field and is then associated to a specific artifact id by the describes_artifact field. The color field in the color database table references a color id in the color_type table. Like the manufacturing_process and material database tables, multiple colors can be associated with a single artifact.

Table 12 – Color table description

color				
field name	data type	mandatory	default value	key type
id	serial	yes	N/A	primary
describes_artifact	int	yes	N/A	foreign
color	int	yes	N/A	foreign

The parameter database table description, see Table 13, is slightly more complex than the material, manufacturing_process, and color tables in the repository. The complexity is because the table exists to record several different types of information. Fields in the parameter database table begin with a serial id and a reference to a specific artifact id number. Additional fields in the table are parameter_type, parameter_metric_type, and parameter_value. The parameter_value field is where an actual numerical value for a specific parameter is recorded. When specifying a parameter value it is not only mandatory to specify the type of parameter but also the metric used to measure the specific parameter.

The parameter_type and parameter_metric_type fields both reference specific tables. A parameter type can be classified as a measurement, which includes descriptors of length, width, height, diameter, etc. The parameter table is also used to store artifact cost, where the type of 'cost' is denoted. Once a specific parameter type is recorded it is necessary to also record the associated parameter type metric. For the examples of measurement the metric may include inches, feet, centimeters, etc. Examples of the cost parameter metric type include US dollars, Canadian dollars, Japanese Yen, etc.

Table 13 – Parameter table description

parameter				
field name	data type	mandatory	default value	key type
id	serial	yes	N/A	primary
describes_artifact	int	yes	N/A	foreign
parameter_type	int	yes	N/A	foreign
parameter_metric_type	int	yes	N/A	foreign
parameter_value	float	yes	N/A	N/A

4.5 Performance-related design knowledge

The performance characteristics of the product are captured next (see Table 14). These are intended to be 'high level' characteristics that describe the overall functionality of the entire product. The performance characteristics expand the component level representation of functionality and translate product functionality into measurable quantities. Designers first specify the type of performance (e.g., weight) and then metrics that are used to define it (e.g., lbs). The characteristic can be specified to occur inherently, at an input, or at an output. If the performance characteristic is tied to a particular component within the product, then the designer can specify this association using the component basis [22]. Finally, the value of the performance characteristic itself must be specified.

The performance_characteristics table begins with the same unique primary key and artifact id referencing. The performance_type, performance_metric_type, and characteristic_type fields all reference their named type database tables. The type database tables are used to store their specific taxonomies. For cases when the performance of a specific component of a more generic artifact is described, the component_basis_type field is used to reference the component basis taxonomy. An example of when the additional component basis specification is needed would be when a motor (the higher level artifact) is recorded, but a torque rating for the output shaft is specified. The performance characteristic will be tied to the motor but more specifically to the output shaft. Following the motor example, electrical requirements could be specific as a certain number of volts for the motor but then also linked to the wire connectors on the motor.

Table 14 – Performance characteristics table description

performance_characteristics				
field name	data type	mandatory	default value	key type
id	serial	yes	N/A	primary
describes_artifact	int	yes	N/A	foreign
performance_type	int	yes	N/A	foreign
performance_metric_type	int	yes	N/A	foreign
characteristic_type	int	no	N/A	foreign
component_basis_type	int	no	N/A	foreign
performance_value	float	yes	N/A	N/A

Customer needs are categorized along with performance characteristics. Although customer needs do not always match actual performance metrics, it is important to denote the desired performance or function when information is available. The customer_needs table description, shown in Table 15, resembles the failure table description in that occurrences, sample_size and rate are specified fields. The customer_need_type table allows for a list of unique customer needs to be established and is referenced by the customer_need field. Importance is also recorded in the customer_needs table. When importance, occurrence, sample size and/or rate is specified, it is possible to evaluate product functionality and performance versus customer needs specifications [32, 33].

Table 15 – Customer needs table description

customer_needs				
field name	data type	mandatory	default value	key type
id	serial	yes	N/A	primary
describes_artifact	int	yes	N/A	foreign
customer_need	int	yes	N/A	foreign
importance	int	no	N/A	N/A
occurrences	int	no	N/A	N/A
sample_size	int	no	N/A	N/A
rate	float	no	N/A	N/A

4.6 Sensory-related design knowledge

The sensory data (see Table 16) is intended to capture additional product data related to the five senses. Finish relates additional data to sight (in addition to previous data organized as color and material). The finish is used to define the visual sheen or luster that covers the largest area of the product and is typically one of three options: brilliant, glossy, or dull. Texture relates to touch and specifies the feel of the product when held and can be one of three options—smooth, rough, or coarse—based on how texture is frequently defined². The dormant and operational smell of the product can also be specified using options such as strong, mild, or none. While smell may not be of concern in many products, some designers pay close attention to the smell of their product such as in the automotive industry when interior smell is an important aspect of their product offering. Designers may also be concerned about how their product tastes (e.g., an electro-mechanical toothbrush); hence, gustatory (taste) data can be included and specified using the four human tastes³: salty, sour, sweet, and bitter. Finally, the operational sound of the product can be specified using general terms such as loud, quiet, etc., or specific dB ratings. Note that the dormant (non-operational) sound is assumed to be quiet and therefore not included in the sensory data.

Table 16 – Sensory table description

sensory				
field name	data type	mandatory	default value	key type
id	serial	yes	N/A	primary
describes_artifact	int	yes	N/A	foreign
finish	int	no	N/A	foreign
texture	int	no	N/A	foreign
dormant_smell	int	no	N/A	foreign
operational_smell	int	no	N/A	foreign
gustatorial	int	no	N/A	foreign
operational_sound	int	no	N/A	foreign

The only mandatory values that are required when sensory data is recorded are the id number and the artifact id reference. After this, the user can specify any sensory items that are of relevance for a particular product. All of the sensory items reference type tables containing their specific taxonomies.

4.7 Media-related design information

There are several types of media that can be associated with artifacts. Media types can take the form of pictures, graphical functional models, graphical assembly models, 2D-CAD files, 3D-CAD files, stereo lithographic (.stl) files for rapid prototyping machines, and

many others. Note that all media types are stored as large objects (files) within the database, thus any associated metadata is also stored. All the types of media, mentioned and unmentioned, reside in the media table of the repository shown in Table 17. Instances of media are unique and associated with an artifact, which is demonstrated by the id and describes_artifact field. The data field in the media table is the “large object” pointer for the actual media files of any type.

Although a single media database table is used to hold all types of media for all of the artifacts, the media_type field allows for specification of the exact media type. The media_type field references an id number in the media_type table. Examples of media types in the media_type table include .jpg, .gif, .stl, .dxf, and .pdf. Having a type associated with a specific piece of media directs the repository software components how to handle and display a piece of media. In cases of .jpg and .gif the repository web site will simply display the image. For .stl, .dxf, and .pdf, the repository web site shows a link for file download and viewing in another application.

Table 17 – Media table description

media				
field name	data type	mandatory	default value	key type
id	serial	yes	N/A	primary
describes_artifact	int	yes	N/A	foreign
media_type	int	yes	N/A	foreign
data	large object	yes	N/A	N/A

5 CONNECTING THE PIECES

As discussed in the previous sections, the design repository consists of numerous data tables to store design information and relationships. The operational design repository contains 41 data tables. The 41 data tables do not include tables that are used to control user access, system authentication and other bookkeeping information. All of the operational design tables and their top level referencing is described and shown in this section. Along with the high-level review of the data schema, features that are enabled by the particular data schema implementation are briefly discussed.

5.1 A high level look at database tables

To begin this section, the database tables are broken up into two separate categories: (1) those that directly store product information and (2) those that are referenced by product storing database tables. There are 13 database tables that directly store product information and 28 supporting tables. All of the 13 product information storing tables were discussed throughout Section 4.

Figure 2 shows a snippet of the design repository schema (Figure 1) with sample data. The boxes represent data tables and arrows represent data relationships. The sample data shown represents only a small subset of design information that can be associated with an artifact. For this example an “electric motor” is shown as an artifact along with associated media, functionality and failure information. The arrows that connect the function_flow, failure, and media tables to the artifact table establish a relationship to the artifact “electric motor.” The corresponding _type table relationships are also shown. For example, looking at the failure table, the failure is denoted as “34” and corresponds to an entry in the failure_type table as being “thermal fatigue.”

All 41 of the repository data tables are represented in Figure 1 with the 13 data storing tables highlighted. A data table makes a reference to another table by an outbound arrow to a particular data table. Looking at the Fig. 1, the failure table references the artifact, failure_type and failure_rating_type tables but is referenced by the failure_data_info table. Tables 18 and 19 show a textual version of Fig. 1. The product information storage tables are listed in Table 18 while the product information support tables are listed in Table 19. While most of these tables were referenced in Section 4, some of them were not directly discussed. Unlike Table 18, Table 19 lists what the particular

² <http://www.freesearch.co.uk/dictionary/texture>, ³ <http://users.rcn.com/jkimball.ma.ultranet/BiologyPages/T/Taste.html>

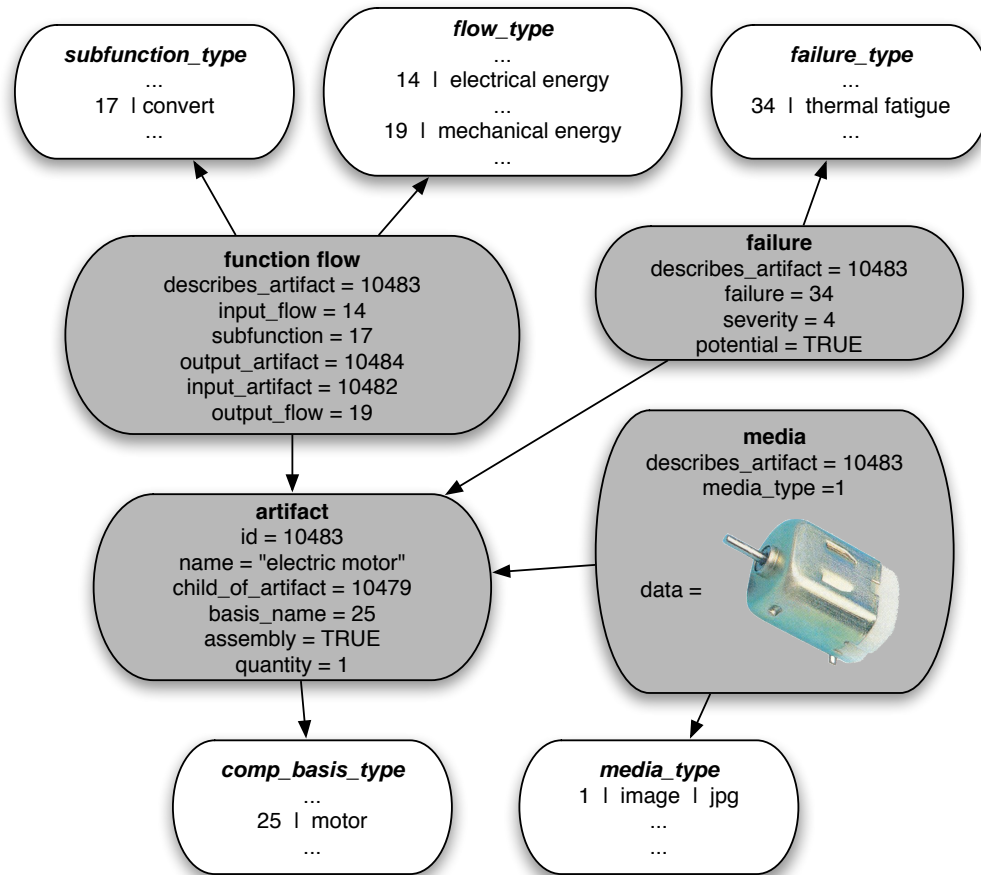


Figure 2 - Repository schema snippet with sample data

Table 18 – Database table listing

	Database Table Name	Description	References
1	artifact	Used to record high level artifact information such as name, description, quantity	system id, creator info type id, comp basis type id, artifact id
2	color	Used to record the color of an artifact	artifact id, color type id
3	customer needs	Used to record customer needs, importance and the number of occurrences	artifact id, customer needs type id
4	failure	Used to record artifact failure modes, severity and likelihood	artifact id, failure type id, failure rating id
5	failure data info	Used to record additional artifact failure information such as data source and the operational environment	failure id, failure data source type id, oper env type id
6	function flow	Used to record artifact functionality	artifact id, subfunction type id, flow type id
7	manufacturing process	Used to record manufacturing process associated with an artifact	artifact id, manufacturing process type id
8	material	Used to record the material of an artifact	artifact id, material type id
9	media	Used to store media such as photos, cad drawings and functional models of artifacts	artifact id, media type id
10	parameter	Used to record physical measurements and artifact cost	artifact id, parameter type id, parameter metric type id
11	performance characteristics	Used to record performance data such as voltage requirement and output torque	artifact id, performance type id, performance metric type id, characteristic type id, comp basis type
12	sensory	Used to record items relating to the five sense such as sound, sight, etc.	comp basis type id, texture type id, smell type id, finish type id, sound type id, gustatory type id, artifact id
13	system	Used to establish a unique product in the repository	system type id, institution type id

Table 19 – Database type table listing

	Database Table Name	Description	Referenced By
1	characteristic type	Used to store the type of performance characteristic (input, output, inherent)	performance characteristics
2	color type	Used to store a list of colors	color
3	comp basis type	Used to store the taxonomy of general components	artifact, performance characteristics, component basis corr type
4	component basis corr type	Used to store synonyms to component basis terms	
5	creator info type	Used to store information about an individual who creates a set of product information	artifact
6	customer needs type	Used to store a list of typical customer needs	customer needs
7	failure data source type	Used to store the list of failure data sources	failure data info
8	failure rating type	Used to store the list of different failure rating scales and conversion factors	failure
9	failure type	Used to store the electrical and mechanical failure taxonomies	failure
10	finish type	Used to store a list of possible artifact finishes	sensory
11	flow corresp type	Used to store synonyms of the flow words of the functional basis	flow type
12	flow type	Used to store the flow words of the functional basis	
13	func corresp type	Used to store synonyms of the function words of the functional basis	
14	gustatory type	Used to store a list of possible artifact tastes	sensory
15	institution type	Used to store a list of types of institutions (academic, industry, etc.)	system
16	manufacturing process type	Used to store a list of manufacturing processes	manufacturing process
17	material type	Used to store a list of material types	material
18	media type	Used to store a list of possible types of media associations and their required actions	media
19	oper env type	Used to store a list of possible artifact operating environments	failure data info
20	parameter metric type	Used to store metrics associated with physical artifact parameters (feet, inches, etc.)	parameter
21	parameter type	Used to store a list of possible types of physical parameters (length, width, etc.)	parameter
22	performance metric type	Used to store units for possible performance parameters (dBA, ft-lbs etc.)	performance characteristics
23	performance type	Used to store a list of possible types of performance parameters (torque, power, etc.)	performance characteristics
24	smell type	Used to store a list of possible types of artifact odors	sensory
25	sound type	Used to store a list of possible artifact sounds	sensory
26	subfunction type	Used to store the function words of the functional basis	function flow
27	system type	Used to store a list of possible system types (consumer, industrial, etc.)	system
28	texture type	Used to store a list of possible artifact textures	sensory

table is referenced by. A majority of the tables listed in Table 18 are referenced by the data tables listed in Table 19; however, some of the supporting tables are referenced by other supporting tables.

5.2 Repository features enabled by the database schema

As mentioned in Section 4, the parent child relationship in the artifact table allows for an artifact hierarchy to be easily established. This hierarchy identifies the as-manufactured assembly/sub-assembly/component relationships as well as supports product architecture representations [33-38]. The same parent child relationship is also im-

plemented in the flow_type and subfunction_type tables. The hierarchy established is between the primary, secondary, and tertiary terms of the function and flow language of the Functional Basis. Because of this hierarchy, the fidelity of product information can be specified during repository browsing, searching, and artifact query reports. For example, when artifact data is specified in the tertiary level of the Functional Basis, the repository can return search results in the secondary level of the Functional Basis. Likewise, when a secondary term of the Functional Basis is specified as a search operator, the repository can return tertiary related artifacts along with the corresponding secondary level relevant artifacts. Although the current component basis

is not hierarchical in nature, these same capabilities have been built into the component_basis table.

Currently the Functional Basis also specifies correspondents to function and flow terms. Correspondents can be thought of as synonyms to the Basis terms. Support has been built into the repository to expand the use of correspondents and feed this information to repository-connected applications. This feature is intended for persons not familiar with the functional basis and automatically tries to associate common language with the appropriate Functional Basis term. As with the hierarchical features, the correspondent and synonym matching support has been built into the Component Basis tables.

6 COMPARISON TO THE ORIGINAL NIST INITIATIVE DATA SCHEMA

The NIST design repository initiative schema takes the form of data structures grouped by classes. This is different than the UMR design repository implementation of data tables and references. Although the implementations take different form, comparisons can still be made to the types of design information each repository specifies.

The NIST schema is broken up into several classes, most of which are referenced directly by the artifact class [1419]. For brevity, we will not present the NIST-defined Abstract classes. The Abstract classes are generally implementation-specific and do not directly relate to the design information allowed by the schema. A summary of the artifact class is shown in Table 20. At first glance the NIST artifact class has a few similarities to the UMR artifact table description (see Table 1). Similar items in Tables 1 and 20 include the artifact name, information/description, type/basis name, and subartifacts of/child of artifact fields. All of the similar fields relate to artifact naming, generic naming (components of artifact family type), and parent-child relationships. When comparing the UMR and NIST schemas, the biggest distinguishing difference is that the UMR schema is singly linked while the NIST schema is doubly linked. Singly linked means that relationships are only drawn in one direction. For example, child_of artifact specifies a parent. Doubly linked implies that for a given item they are linked to both parents and children. This is noticeable in the NIST artifact class in the ‘subartifacts’ and ‘subartifacts of’ descriptors. There are tradeoffs associated with both singly and doubly linked implementations of data structures. Singly linked items usually require more computational overhead to perform complex queries while doubly linked items require additional storage space and are often more difficult to maintain. The UMR schema implements singly linked lists to reduce the database size and storage requirements.

The NIST schema specifies placeholders for function and flow in the ‘function,’ ‘is source of,’ and ‘is destination of’ references. Although the implementation takes different form than the UMR repository schema, the same types of relevant design information are being stored. The function and flow definitions in the NIST schema actually reference classes that further outline the structure of information. The ‘form’ class reference under the NIST artifact class also references additional information. Placeholders for design information in NIST’s form class include material, geometry, and form type. NIST’s form class designation relates mainly to UMR’s material and parameter tables. The ‘references’ and ‘is referenced by’ designation in the NIST artifact class are used to establish artifact-to-artifact relationships. The same type of artifact-to-artifact relationships are also present in the UMR schema, but take form in the function_flow table as the input and output artifacts.

The artifact class of the NIST schema also contains a placeholder for ‘is specified by’ which points to a specification class. The specification class allows for information pertaining to requirements to be associated with an artifact. Requirements in the UMR schema mainly take form in the performance_characteristics table. NIST’s ‘is member of’ and ‘is special member of’ class designations relate to the system table in the UMR schema. The ‘behavior’ designation contained in the NIST artifact class refers mainly to the type of mathematical based

Table 20 – NIST artifact class

artifact		
information	mandatory	description
name	yes	Is used to specify the artifact name
information	yes	Captures the name, description, documentation, methods and properties
references	no	Specifies references from this object to another object
is referenced by	no	Specifies references from other objects to this object
is member of	no	Specifies sets of which this artifact is a member
is special member of	no	Specifies directed sets of which this is a member
type	yes	Specifies the artifact family
is specified by	no	Specifies the type of specification
config info	yes	Specifies configuration information
function	yes	Specifies artifact functionality
form	yes	Specifies artifact form
behavior	no	Specifies artifact behavior as a function of input in mathematical form
subartifacts	no	A list of children artifacts
subartifacts of	no	A list of parent artifacts
is source of	no	Specifies a flow of which this is an input artifact
is destination of	no	Specifies a flow of which this is an output artifact

equation that describes a particular artifact’s behavior. There is no real mapping of NIST’s behavior class to the UMR schema; however, some types of behavior could be derived from the sensory and performance_characteristics tables. Not described in NIST’s artifact class is the transfer function class. Transfer functions are not directly recorded in the UMR schema at this time although portions (input and output values) could also be derived from the sensory and performance_characteristics tables.

The UMR schema is distinguished from the NIST schema by allowing design information regarding customer needs, component basis designations, manufacturer, failure modes and sensory level information to be stored. Also, the NIST schema is only exemplary and has not been implemented in a distributable and publicly accessible system. The UMR repository schema is more complete and has been implemented and tested with a large set of design data.

7 CONCLUSIONS

The UMR design repository schema (version 2.0) represents several years of development and has undergone multiple revisions and updates. Although the task of building and expanding systems to digitally represent design information will continue, the schema presented in this paper provides a roadmap to future revisions and supports design information storage support to modern repository connected applications. The UMR Design Repository—and more specifically the types of design information recorded—began with a somewhat limited set of design information: component-to-component connects, function-to-component connections, and basic artifact bills of materials. The repository schema has expanded on the initial data set to include failure and risk-based information, generic component naming, function and flow hierarchies, multiple types of media associations, and sensory-related artifact descriptions.

A comparison of the UMR design repository schema and the NIST schema shows a great deal of overlap. Because of the specific implementation and data classifications used at UMR, research in the area of transfer function and artifact behavior remains an ongoing project. The UMR schema does, however, expand on the NIST schema by specifying data placeholders for failure, sensory, performance characteristics, customer needs, and media artifact information. By implementing strict data relationships and taxonomy/basis lists, the UMR schema enforces design information consistency along with artifact representation and structure. The PostgreSQL database and underlying schema provides a robust and extensible platform for design information storage and future schema development.

Future work of the UMR design repository schema includes the integration of mathematical-based transfer functions, function and flow synonym lookup implementation, Component Basis synonym lookup implementation, and further database optimization and analysis. In order to allow for mathematical-based performance analysis of

systems and subsystems it is necessary to build repository infrastructure to store the supporting data and relationships. The difficult task associated with including transfer functions is the classification of the associated mathematical formulas and maintaining database relational integrity when tracing mathematical and numerical variables. The most visible starting point would be to record simplified algebraic equations in order to reduce calculation complexity and to develop the required variable storage infrastructure.

To implement the synonym lookup feature that is supported by the current repository schema for the function, flow, and component basis tables involves similar tasks, namely, developing the list of relevant synonyms. The relevant synonyms will most likely be found by first using a standard thesaurus and dictionary to pick out commonly associated keywords. A more detailed experiment may be formulated to interview inexperienced designers and engineers and then translate their comments to the appropriate basis terms. These efforts will also examine the use of ontologies to facilitate function and component matching across artifacts [34].

Two main metrics to measure database performance are cohesion and coupling. Several tools exist that can perform such measurements and then provide suggestions for improving database performance and integrity. Other optimizations may take the form of moving one or two data types between tables. Although regrouping data types is not expected, it may make sense in the future to do so. Note that further optimization and analysis would only alter the specific implementation and not affect the types of design information that are recorded.

ACKNOWLEDGEMENTS

This work is supported by the National Science Foundation under Grant Nos. IIS-0325415 and IIS-0325402. Any opinions or findings of this work are the responsibility of the authors and do not necessarily reflect the view of the National Science Foundation or our collaborators.

REFERENCES

- [1] Bohm, M., R. Stone and S. Szykman, 2005, "Enhancing Virtual Product Representations for Advanced Design Repository Systems," *Journal of Computer and Information Science in Engineering*, 5 (4): 360-372.
- [2] Bryant, C., D. McAdams, R. Stone, T. Kurtoglu and M. Campbell, 2005, "A Computational Technique for Concept Generation," *Proceedings of IDETC/CIE 2005*, DETC2005-85323, Long Beach, CA.
- [3] Bryant, C., R. Stone, D. McAdams, T. Kurtoglu and M. Campbell, 2005, "Concept Generation from the Functional Basis of Design," *International Conference on Engineering Design, ICED 05*, Melbourne, Australia.
- [4] Shooter, S. B., T. W. Simpson, S. R. T. Kumaray, R. B. Stone and J. P. Terpenny, 2005, "Toward a Multi-Agent Information Infrastructure for Product Family Planning and Mass Customization," *International Journal of Mass Customization*, 1 (1): 134-155.
- [5] Bohm, M., R. Stone and S. Szykman, 2003, "Enhancing Virtual Product Representations for Advanced Design Repository Systems," *Proceedings of DETC2003*, DETC2003/CIE-48239, Chicago, IL.
- [6] Summers, J., D. Maxwell, C. Camp and A. Butler, 2000, "Features as an Abstraction for designer convenience in the Design of Complex Products," *DETC*, DETC200/CIE-14642, Baltimore, MD.
- [7] Dixon, J., E. Libardi, S. Luby, M. Vaghul and M. Simmons, 1987, "Expert Systems for Mechanical Design: Examples of Symbolic Representations of Design Geometries," *Engineering Computing*, 2 (1): 1-10.
- [8] Cherneff, J., 1991, "Communicating design representations: the role of interpretation," *Computer-Aided Engineering Journal*, 8 (4): 153-159.
- [9] Sturges, R., K. O'Shaughnessy and M. Kilani, 1996, "Computational model for conceptual design based on extended function logic," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 10 (4): 255-274.
- [10] Regli, W. C. and D. M. Gaines, 1997, "A repository for design, process planning and assembly," *Computer-Aided Design*, 29 (12): 895-905.
- [11] Terpenny, J. P. and M. Benson, 2001, "A Survey of Methods and Approaches to Knowledge Management in the Product Development Environment," *Proceedings of the ASME 2001 Design Engineering Technical Conference*, Pittsburgh, PA.
- [12] Candy, L., 1998, "Representations of strategic knowledge in design," *Knowledge-based Systems*, 11 (7-8): 370-390.
- [13] Svensson, D. and J. Malmqvist, 2001, "Integration of Requirement Management and Product Data Management Systems," *DETC*, DETC2001/CIE-21246, Pittsburgh, PA.
- [14] Szykman, S., R. Sriram and S. Smith, 1996, *Proceedings of the NIST Design Repository Workshop*, National Institute of Standards and Technology, Gaithersburg, MD.
- [15] Murdock, J., S. Szykman and R. Sriram, 1997, "An Information Modeling Framework to Support Design Databases and Repositories," *Proceedings of DETC'97*, DETC97/DFM-4373, Sacramento, CA.
- [16] Szykman, S., J. Racz and R. Sriram, 1999, "The Representation of Function in Computer-Based Design," *Proceedings of the ASME Design Theory and Methodology Conference*, DETC99/DTM-8742, Las Vegas, NV.
- [17] Shooter, S., W. Keirouz, S. Szykman and S. Fennes, 2000, "A Model For Information Flow In Design," *Proceedings of the ASME Design Theory and Methodology Conference*, DETC2000/DTM-14550, Baltimore, MD.
- [18] Szykman, S., S. Fennes, S. Shooter and W. Keirouz, 2001, "A Foundation for Interoperability in the Next-Generation Product Development Systems," *Computer-Aided Design*, 33 (7): 545-559.
- [19] Szykman, S., 2002, "Architecture and Implementation of a Design Repository System," *Proceedings of DETC2002*, DETC2002/CIE-34463, Montreal, Canada.
- [20] Douglas, K. and S. Douglas, 2003, *PostgreSQL*, SAMS Publishing.
- [21] Hirtz, J., R. Stone, D. McAdams, S. Szykman and K. Wood, 2002, "A Functional Basis for Engineering Design: Reconciling and Evolving Previous Efforts," *Research in Engineering Design*, 13 (2): 65-82.
- [22] Kurtoglu, T., M. Campbell, C. Bryant, R. Stone and D. McAdams, 2005, "Deriving a Component Basis for Computational Functional Synthesis," *International Conference on Engineering Design, ICED'05*, Melbourne, Australia.
- [23] Stone, R., I. Tumer and M. Stock, 2005, "Comparing Two Levels of Functional Detail for Mapping Historical Failures: You Are Only as Good as Your Knowledge Base," *Research in Engineering Design*, In Press, DOI 10.1007/s00163-005-0005-z.
- [24] Stone, R., I. Tumer and M. Van Wie, 2004, "The Function Failure Design Method," *Journal of Mechanical Design*, 127 (3): 397-407.

- [25] Grantham Lough, K., R. Stone and I. Tumer, 2005, "Function Based Risk Assessment: Mapping Function to Likelihood," *Proceedings of the ASME International Design Engineering Technical Conference*, Long Beach, CA.
- [26] Bohm, M. and R. Stone, 2004, "Representing Product Functionality to Support Reuse: Conceptual and Supporting Functions," *Proceedings of DETC2004*, DETC2004-57693, Salt Lake City, UT.
- [27] Lock, D., 1993, *Handbook of Engineering Management*, Butterworth-Heinemann, Oxford.
- [28] Tumer, I. Y. and R. B. Stone, 2003, "Mapping Function to Failure During High-Risk Component Development," *Research in Engineering Design*, 14 (1): 25-33.
- [29] Tumer, I. Y. and R. B. Stone, 2003, "Analytical Methods for Mapping Function to Failure During High-Risk Component Development," *Research in Engineering Design*, 14 (1): 25-33.
- [30] Tumer, I. Y., R. B. Stone and D. G. Bell, 2003, "Requirements for a Failure Mode Taxonomy for Use in Conceptual Design," *International Conference on Engineering Design*, Stockholm Sweden.
- [31] Tumer, I. Y., R. Stone, R. A. Roberts and A. F. Brown, 2003, "A Function-Based Exploration of JPL's Problem/Failure Reporting Database," *ASME International Mechanical Engineering Congress and Expo*, IMECE2003-42769, Washington, D.C.
- [32] McAdams, D., R. B. Stone and K. L. Wood, 1999, "Functional interdependence and product similarity based on customer needs," *Research in Engineering Design*, 11 (1).
- [33] Yu, J. S., J. P. Gonzalez-Zugasti and K. N. Otto, 1999, "Product Architecture Based Upon Customer Demand," *ASME Journal of Mechanical Design*, 121 (3): 329-335.
- [34] Nanda, J., T. W. Simpson, S. R. T. Kumara and S. B. Shooter, 2006, "Product Family Ontology Development Using Formal Concept Analysis and Web Ontology Language," Accepted to *ASME Journal of Computing and Information Science in Engineering*.
- [35] Hernandez, G., T. Simpson, J. Allen, E. Bascaran, L. Avila and F. Salinas, 2001, "Robust Design of Product Families with Production Modeling and Evaluation," *Journal of Mechanical Design*, 123 (2): 183-190.
- [36] Shooter, S. B., T. W. Simpson, S. R. T. Kumara, R. B. Stone and J. P. Terpenney, 2004, "Toward an Information Management Infrastructure for Product Family Planning and Platform Customization," *ASME Design Engineering Technical Conferences*, DETC2004/DAC-57430, Salt Lake City, UT.
- [37] Zamirowski, E. J. and K. N. Otto, 1999, "Identifying Product Portfolio Architecture Modularity Using Function and Variety Heuristics," *ASME Design Engineering Technology Conference, Design Theory and Methodology Conference*, DETC99/DTM-8760, Las Vegas, NV.
- [38] Otto, K. and K. Wood, 1997, Conceptual and Configuration Design of Products and Assemblies, *ASM Handbook, Materials Selection and Design*, ASM International.