



Contents lists available at ScienceDirect

Computer-Aided Design

journal homepage: www.elsevier.com/locate/cad

Introduction of a data schema to support a design repository

Matt R. Bohm^{a,*}, Robert B. Stone^a, Timothy W. Simpson, Elizabeth D. Steva^a Design Engineering Laboratory, Department of Interdisciplinary Engineering, Missouri University of Science and Technology (formerly University of Missouri - Rolla), MO 65409, United States^b Engineering Design Optimization Group, Department of Mechanical & Nuclear Engineering, The Pennsylvania State University, University Park, PA 16802, United States

ARTICLE INFO

Article history:

Received 8 August 2006

Accepted 2 September 2007

Keywords:

Design repository schema

Conceptual design

ABSTRACT

This paper presents the data schema required to capture fundamental elements of design information in a heterogeneous repository supporting design reuse. Design information captured by the repository can be divided into seven main categories of artifact-, function-, failure-, physical-, performance-, sensory- and media-related information types. Each of the seven types of design information is described in detail. The repository schema is specific to a relational database system driving the implemented design repository; however, the types of design information recorded are applicable to any implementation of a design repository. The aim of this paper is to fully describe the data schema such that it could be recreated or specialized for industrial or research applications. The result is a complete description of fundamental design knowledge to support design reuse and a data schema specification. The data schema has been vetted with the implemented design repository that contains design information for over 100 consumer electro-mechanical products.

© 2007 Elsevier Ltd. All rights reserved.

1. Introduction

The objective of a design repository is to allow designers to store and retrieve design knowledge at various levels of abstraction – from form (components, sub-assemblies and assemblies as well as historical performance and failure data) to architecture to function. The different levels of abstraction and types of design information provide innovative ways to approach design. A design by analogy approach, for example, uses a functional or product architecture description to find other existing products which are similar to it, thus providing a starting point for a form solution. A risk conscious approach, for example, uses conceptual level functionality to find related failure information to determine values for risk likelihood and consequence. With a well populated repository, emerging concept generator algorithms take, as input, basic product functionality and synthesize, filter and rank concepts to use as baselines for further product development. While the possibilities design repositories offer are diverse and helpful to designers, the implementation of such repositories are crucial to their overall success and usefulness.

Realizing the potential impact of an operational design repository, researchers at UMR, the University of Texas at Austin and the National Institute of Standards and Technology (NIST)

began gathering artifact information in 1999 [1]. Since that time, the process in which artifact data is gathered and recorded has changed significantly. Initially, artifact design information was recorded in spreadsheets and mainly took the form of Bills of Materials (BOM), Function Component Matrices (FCM), and Design Structure Matrices (DSM). While this type of information was useful, it was also limited in scope and the required manipulations to compute with the data were cumbersome. A prior Design Repository Project initiative by NIST helped to guide the design repository project hosted at UMR to a more mature state. To enhance data integrity, design information was migrated from various independent file formats to a relational database. A web-based repository navigator including search and design tool generation features was created along with a repository entry application creating the user interface for the new repository, dubbed the UMR Design Repository (also referred to as simply the Design Repository in this article).

More recently, UMR has further partnered with UT-Austin [2, 3], Penn State, Virginia Tech and Bucknell [4] to expand the types of design information and breadth of design tool features within the repository. Currently, the Design Repository serves as a hub for designers for information exchange and design generation tools. Information entry and retrieval occurs within a standalone application (available at <http://function.basiceng.umn.edu/repositoryEntry>) while information retrieval occurs over the Internet through the Design Repository's web portal (<http://function.basiceng.umn.edu/repository>). The infrastructure supporting these two applications is the Design Repository database and more specifically the database schema. The database schema establishes

* Corresponding author. Tel.: +1 573 341 4606; fax: +1 573 341 6593.

E-mail addresses: mbohmm@mst.edu (M.R. Bohm), rstone@mst.edu (R.B. Stone), tw8@psu.edu (T.W. Simpson), eds165@psu.edu (E.D. Steva).

what types of design information can be stored, the relationship of those elements and the extensibility of including new and additional types of design information.

The objective of this paper is to fully describe the database schema, currently at version 2.0, powering the repository. 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 4 provide implementation level details, Section 5 presents the larger context of repository operations and Section 6 summarizes conclusions prior to future work in Section 7.

2. Background

Several types of applications have been created to record pieces of product or process information. This niche of design-based applications includes the NIST Repository initiative, PDM (Product Data Management) systems, and CAD-based repositories. The NIST Repository Initiative set forth guidelines for categorizing function-centric design information. PDM systems allow for part hierarchy storage as well as process data and project management elements. CAD-based design repositories store numerous artifact CAD files and rely on feature descriptions and recognition capabilities. In this section an overview of the NIST Repository Initiative, PDM, and CAD-based storing systems is presented.

2.1. NIST Repository Initiative

The most similar system to UMR's repository schema takes form in the NIST Design Repository representation model [5–10]. Through their repository initiative, NIST set out to define basic guidelines of a design repository and how archived design information could be useful to designers. 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. While portions of the NIST initiative overlaps with design representation standards such as STEP (Standard for the Exchange of Product model data), the breadth and scope of implementation differ greatly. Like certain STEP protocols, the NIST framework provides for geometric and process information storage but also expands them to a higher-level domain of design information storage.

The NIST initiative proposes a set of information models to be used for modeling product knowledge at varying levels of detail. There are several data entities which allow for a variety of aspects of a product description to be represented. The classes specified in the NIST Core Product Model include: Artifact, Function, Transfer Function, Flow, Form, Geometry, Material, Behavior, Specification, Configuration, Relationship, Requirement, Reference and Constraint.

The UMR schema contains elements similar to that of the NIST schema but is distinguished 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, publicly accessible and operational system.

2.2. PDM 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 [11] 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, similar pieces of design knowledge, such as CAD models and part hierarchies, are stored; however, the main focus is the mapping between functions and components and the compatibility of components to connect together as a system.

2.3. CAD-based systems

Regli [12], 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.

3. UMR Design Repository conventions

The UMR Design Repository is an artifact-centric repository, meaning that for a design attribute to exist, it must be linked to an artifact. Other information classes 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.

The repository schema is built and served by a PostgreSQL (an SQL variant) database [13]. 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 table: (1) a database table description, which details 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 details the set of data entries in the database (alternatively, the actual data that is entered into the repository – in this case, the product data). Note, every database table has an associated database table description. The term row will be used to describe an entry in a database table and the term field will be used to describe an object in the database table description. Table 1 shows the artifact table description in the repository database schema.

All table descriptions throughout this paper are 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

Table 1
Artifact table description

Field name	Data type	Mandatory	Default value	Key type
Id	Serial	Yes	N/A	Primary
Name	Varchar	Yes	N/A	N/A
Child_of_artifact	Int	No	N/A	Foreign
Basis_name	Int	No	N/A	Foreign
Serial_id_number	Varchar	No	N/A	N/A
Assembly	Boolean	Yes	FALSE	N/A
Description	Varchar	No	N/A	N/A
Quantity	Int	Yes	1	N/A
System	Int	Yes	N/A	Foreign
Manufacturer	Varchar	No	N/A	N/A
Trademark	Varchar	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

particular field. Only one primary key can exist per table and is used to develop a unique reference to the particular entry in that database table. Having a foreign key designation means that the particular field references the primary key of another database table. All database 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 database table's primary key.

Within the UMR Design Repository, there are two main categories of table—those that store artifact-specific design data information and those that store taxonomies and bases to classify design information. The Design Repository makes use of several taxonomies and bases to describe information such as functionality, failure modes, manufacturing processes, materials and color. While several different taxonomies exists to describe

Table 2
Subfunction_type table description

Field name	Data type	Mandatory	Default value	Key type
Id	Serial	Yes	N/A	Primary
Subfunction	Varchar	Yes	N/A	Foreign
Tier	Int	Yes	N/A	N/A
Child_of_subfunction	Int	No	N/A	Foreign
Definition	Varchar	No	N/A	N/A

these types of information, ones chosen for use within the repository could alternatively be replaced for specific repository implementations. The tables that store taxonomies and bases are denoted with _type after the table name. Fig. 1 shows all 41 of the repository database tables with the 13 database 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 database table: the subfunction_type table description.

As with all other database table descriptions in the repository, the subfunction_type table begins with a serial id that establishes a primary key. The second field of the database 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 [14]. 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.

4. UMR Design Repository data groups

The design information captured by the Design Repository data schema can be broken up into seven main classes: artifact-,

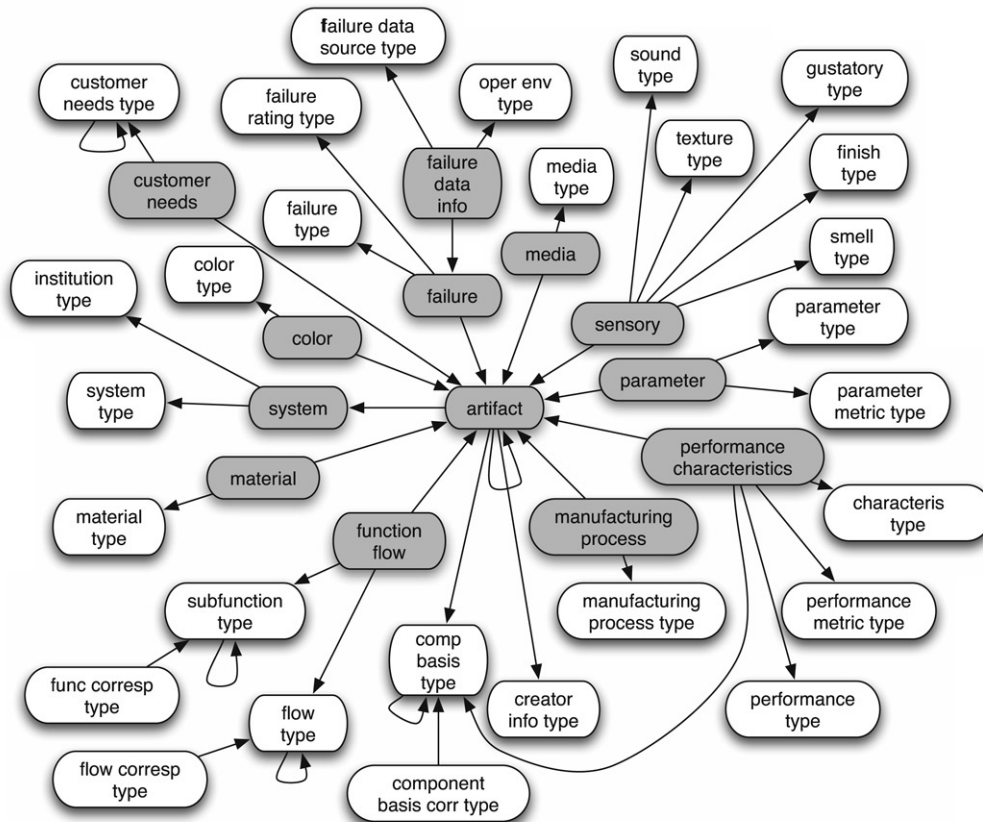


Fig. 1. Graphical view of repository database tables.

Table 3
System table description

System				
Field name	Data type	Mandatory	Default value	Key type
Id	Serial	Yes	N/A	Primary
Name	Varchar	Yes	N/A	N/A
System_type	Int	Yes	N/A	Foreign
Description	Varchar	No	N/A	N/A
Contributing_institution	Varchar	Yes	N/A	N/A

function-, failure-, physical-, performance-, sensory- and media-related information types. All seven of these categories are represented in different database tables but are all brought together by the use of database table relationships, found in the database table descriptions for each database table. In this section, each of the seven data classes is 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 database table stores directly. Each artifact comprises a row entry 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 of 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 relationships such that a product artifact hierarchy is created. In order to keep a strict separation of different products within the repository a system database table is used; the system database table description is shown in Table 3.

Looking back at the artifact database table description shown in Section 3 (see Table 1), there is a placeholder for a system reference for each artifact instance. A unique system id is established for each new product that is added to the repository. Every artifact belonging to the given system is then referenced to the system id. In the system database table description (see Table 3), a system name, system description, and contributing institution are associated with the system. For example there may be 30 artifacts named ‘motor’ that are unique to different products because of the system designation. The contributing_institution field in the system database table is used to track 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, commercial, automotive, space, etc.

The artifact database table description (see Table 1) begins with a serial-based id number to establish a unique serial number for each artifact that resides in the database. Moving through the artifact table, data fields such as the artifact name, description, quantity, manufacturer, trademark, artifact release date, entry date and modification date are present. The child_of_artifact field 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.

Next in the artifact database table, the basis_name field is used to associate a component basis name to a specific artifact [15]. For example an artifact denoted as a coffee cup would reference

Table 4
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

the component_basis_type table to establish that ‘reservoir’ is the corresponding component basis term. Component basis naming is used to cluster similar artifacts. 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 bookkeeping purposes, the creator_info field is used. The creator_info field references the creator_info_type database table, which contains contributor information such as their name, email address, and affiliation.

4.2. Function-related design knowledge

Product functionality is highly important not only to conceptual design but also to other design and optimization methods that use function as a link to existing design information. 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 database 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 4 shows the function_flow database table description.

Similar to the artifact database table, the function_flow database table begins with a serial id number creating a unique primary key. The primary key ensures that each set of function and flow descriptions is 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 [16]. A conceptual function is a function that is required by customer needs where supporting functions describe the necessary functions required for the physical embodiment of the product. 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. 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

Table 5

Function flow database table with sample data

Function_flow							
Id	Describes	Input_artifact	Input_flow	Subfunction	Output_flow	Output_artifact	Supporting
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 6

Function flow database table with translated sample

Function_flow							
Id	Describes	Input_artifact	Input_flow	Subfunction	Output_flow	Output_artifact	Supporting
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

an incoming flow of electrical energy from two specific sources both electrical energy flows would be transferred to the designated artifact with an output_artifact 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. All fields within the function_flow database table are set as mandatory. From a functional perspective, it would not make sense to list a function without also listing the incoming and outgoing flows or the destination.

Table 5 shows the function_flow database table populated with sample data to demonstrate how function relationships are generated. Reading across the table, the 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 5 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 5, 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 version of Table 5 is shown in Table 6. For both Tables 5 and 6, the functions are described as conceptual functions, taking the value of FALSE in the supporting field [16].

4.3. Failure-related design knowledge

Failure information in this Design Repository is driven by efforts including the Function-Failure Design Method (FFDM) [17,18], Risk-in-Early Design (RED) [19], and adaptations of modern Failure Modes and Effects Analysis (FMEA) [20] techniques. FFDM and RED are similar in purpose to generic FMEA methods but strive to provide risk and possible failure information at the conceptual level of design based solely on product functionality. 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 [17,18].

Like the function_flow database table description, the failure database table description shown in Table 7 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 the 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 database 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. Actual failure modes are those that have been recorded historically where potential failure modes are those that are believed to be physically possible. Because of this very distinct difference 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 default rating scale assumed in the repository is the 1–5 severity scale [20]. 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

