Proceedings of the
1999 ASME Design Engineering Technical Conferences
September 12-15, 1999, Las Vegas, Nevada

DETC99/CIE-9025

THE USE OF XML FOR DESCRIBING FUNCTIONS AND TAXONOMIES IN COMPUTER-BASED DESIGN

Simon Szykman, Jocelyn Senfaute and Ram D. Sriram
Manufacturing Systems Integration Division
National Institute of Standards and Technology
Building 304, Room 6
100 Bureau Drive, Stop 8262
Gaithersburg, MD 20899-8262

Keywords: Design Repositories, Function, Information Modeling, Representation, Taxonomy, XML

ABSTRACT

A standardized representation of engineering function has been developed, consisting of schemata for functions and associated flows along with taxonomies of generic functions and flows. This paper presents a mapping of this representation into the Extensible Markup Language (XML), a language similar in appearance to HTML but which allows the development of user-defined tags, various kinds of references, and other mechanisms. The formal representation provides the means for representing functions that have multiple input and output flows, properties and parameters associated with these flows, and the decomposition of functions into subfunctions each potentially having its own distinct flows. This mapping has been developed in order to support representation of artifact function models in software systems, as well as to provide a neutral format for exchange of function-based information among software systems.

1 INTRODUCTION

As industry continues to increase its reliance on knowledge and information in the product development process, the capabilities of computer-aided design systems are evolving in response to industry's changing needs. One of the more significant cultural shifts in industry is a trend toward design processes that are more knowledge-based, distributed, and collaborative. The increasing complexity of engineering systems, coupled with the fact that many such systems are no longer developed by a small group of co-located designers within a single company, makes effective capture, retrieval, reuse, and exchange of knowledge a critical issue.

The National Institute of Standards and Technology (NIST), which views U.S. industry as its primary customer, has held numerous design-related industry workshops in order to identify and anticipate new needs. The work presented in this paper has been motivated by input received at one such workshop, the NIST Design Repository Workshop (held in November, 1996), which concerned needs for representations and technology to support more effective use and reuse of knowledge in design. Discussion of the needs associated with representation of engineering function arose in three different breakout sessions. Specific statements indicated (1) a need for representation of function in CAD, in addition to geometry, (2) a need for a fixed representation scheme for modeling function, and (3) a need for a commonly agreed-upon set of functions performed by mechanical systems (Szykman et al., 1998).

Driven by these requirements, an initial specification for a standardized representation of engineering artifact function has been developed that includes schemata (information models) for representation of function and associated flows, as well as an initial attempt at developing taxonomies of functions and flows. The objective of the latter effort is to generate taxonomies that are as small as possible, yet generic enough to allow modeling of a broad variety of engineering artifacts. The result is a formal representation that provides the means for representing functions that have multiple input and output flows, properties and parameters associated with flows, and the decomposition of functions into subfunctions each potentially having its own distinct flows. The representation also provides a mapping from the function domain to the physical domain (via references to artifacts in the flow schema) and supports the representation of

function sharing provided that it is used with an artifact representation that permits artifacts to have multiple functions. This representation is described in detail in (Szykman et al., 1999b).

Capturing information about artifact function is only one step towards realizing an impact on product development. Equally important is the ability to share and exchange knowledge with other individuals, design teams, suppliers, corporate partners, etc., who in practice will often not be using the same software systems. Capture of information is only of limited use if the information cannot be effectively communicated to others. Indeed, a major barrier to information exchange in industry today is the proliferation of incompatible proprietary formats for representation of artifact geometry. An analogous problem could easily inhibit the effective use of function-based representations and models in product development. Once artifact function models are more broadly used by industry, the question of how to exchange information between software systems will become important. Since design knowledge is typically stored in some kind of database rather than in plain text files, the generic schemata and taxonomies introduced in (Szykman et al., 1999b) may not be sufficient to allow exchange of information between software systems.

This paper presents a mapping of the schemata and taxonomies introduced in (Szykman et al., 1999b) into the Extensible Markup Language (XML) (World Wide Web Consortium, 1998), a language similar in appearance to the Hypertext Markup Language (HTML) (World Wide Web Consortium, 1995) but which allows the development of user-defined tags, various kinds of references, and other mechanisms. This mapping has been developed in order to provide a neutral format for exchange of function-based information among software systems. The XML specification imposes guidelines on how to structure a document (data), how to represent schemata, how to make references, and so on. This provides advantages over, say, a plain text file format for artifact function models.

The following section describes related work in this area, as well as additional advantages of XML over other languages that could have been used in its place. Section 3 describes the XML-based schemata for the data structures used to represent artifact function and associated flows. Section 4 discusses the XML representation of the taxonomies of generic functions and flows to be used in association with the function and flow schemata. Section 5 provides a discussion of implementational issues related to information processing and use of this representation for knowledge retrieval and reuse. Areas for future research are discussed in Section 6.

2 RELATED WORK

The use of function has long-since been recognized as an important part of the design process. Formalization of approaches to representing and reasoning about function, and using this knowledge to drive design, are in comparison relatively new in the engineering field. Much of the early research in the area of function representation was performed in the artificial intelligence (AI) field. Even definitions of function have varied, indicating that the concept is a complex one. Common definitions include any of a number of variations on one proposed by Rodenacker (1971), which defines function as a relation between the input and output of energy, material and information. Pahl and Beitz (1988) adapt this characterization but generalize the concept, defining it as an abstract formulation of a task, independent of any particular solution.

The variety in definitions of function have led to a variety of uses and representations of function. Baxter (1994) distinguishes two types of representations: models of flows between inputs and the outputs of products, and syntactic languages. The first type generally follows the Pahl and Beitz paradigm of the flow of materials, energy and signals through a hierarchy of functions. The overall function is determined for a system and then broken down into a set of subfunctions. This type of decomposition yields a functional graph that roughly approximates subassembly boundaries (Shapiro and Voelcker, 1989). Other approaches including (Kirschman and Fadel, 1998), (Umeda and Tomiyama, 1997), and the one taken in this paper, view the functional description of a system as being described by an abstract functional decomposition that may, but need not, have a direct mapping onto an isomorphic physical decomposition of assemblies and subassemblies.

Syntactic languages describe a design artifact using a grammatical approach where a grammar is used to capture information about function. In general, these grammars consist of combinations of verbs (functions) and nouns (parts of a design artifact, or flows) such as “hold liquid” (Lai and Wilson, 1989), “crush material” (Hundal, 1990), “create lateral motion” (Sturges et al., 1996), “transmit linear motion” (Kirschman and Fadel, 1998), and “convert electricity to thermal energy” (Stone and Wood, 1999). Approaches such as these can capture the essence of many artifact functions; however, they do not fully address the needs of a formal representation of function because they do not include information models that capture other types of information relevant to function, or the explicit mappings between functions to flows, and between the function domain and the physical domain. The representation of function described in this paper seeks to address these limitations by providing formal schemata and taxonomies of terms used to describe artifact functions and associated flows.

XML is not the only language that has been developed for information modeling and knowledge exchange; other such languages include EXPRESS (ISO, 1994b), Knowledge Interchange Format (KIF) (Genesereth and Fikes, 1992), and Open Knowledge Base Connectivity (OKBC) (Chahudri et al., 1998). However, XML has the main advantage of increasingly widespread adoption in the information technology world. More specifically, XML support is expected in upcoming versions of several commercial Web (i.e., World Wide Web) browsers and word processing applications, in addition to a number of XML authoring and development tools that are currently available. The decision to use XML has been made in order to provide a more broad-based solution to an industry that is increasingly looking towards purchase of off-the-shelf software over in-house development when possible.

Other examples of the use of XML in engineering design research include (Veeramani et al., 1998), which proposes the use of XML as a modeling language for an intranet decision support system, and (Lee and Hahn, 1998) which uses XML as a basis for a manufacturing process engineering language. Be-

cause XML is a relatively new language—having achieved the pre-standard status of recommendation from the World Wide Web Consortium in February, 1998—it's use in the engineering field has not yet reached the pervasive level that has been seen in the information technology world.

3 REPRESENTATION OF FUNCTION

This section introduces XML-based schemata, or information models, used for the generic representation of function and associated flows. Within the context of this representation, function and flow are represented separately using different schemata. The motivations behind the decision to decouple the representation of function and flow concern ease of modification of an artifact function representation, avoiding a proliferation of concepts required for modeling artifact function, and simplifying the representation of functions that do not have flows associated with them. A more detailed discussion of these issues is given in (Szykman et al., 1999b) and therefore is not repeated here. Similarly, because (Szykman et al., 1999b) contains in-depth explanations of the generic (not XML-based) schemata for function and flow, this paper contains only abbreviated descriptions.

The next section provides a brief introduction to XML for those who are not familiar with the language. The sections that follow describe the mapping of generic schemata for function and flow into XML. An example illustrating the use of the schemata for function modeling of a design artifact is then given. The taxonomies of generic functions and flows will be discussed in Section 4.

3.1 XML: A Brief Introduction

The Extensible Markup Language (XML) is a specification developed by the World Wide Web Consortium (W3C). XML is similar in appearance to the Hypertext Markup Language (HTML). Unlike HTML, which has a fixed set of elements (or *tags*), XML is designed to be extensible by allowing the definition of new elements arbitrarily, or more commonly using a specified format called Document Type Definition (DTD). The ability of XML to allow definition of structures and elements fosters the development of standard schemata and terminologies. User-defined XML elements can be read by any other compliant system, providing a way to keep data in a structured, yet neutral, platform-independent form. Although XML is a structured, machine-readable language, another benefit of XML is that it is also human-interpretable (assuming the human is familiar with the structure of XML). XML can therefore be used either for computer-based data exchange, as well as for human consumption.

XML documents are composed of markup and content. There are six kinds of markup that can occur in an XML document; the only ones discussed here are the two most important ones necessary for understanding how a document is built and structured: elements and document type declaration. Elements are the most common form of markup. Delimited by angled brackets ("< >"), most elements identify the nature of the content they surround. Some elements may be empty, in which case they have no content. If an element is not empty, it begins with a start tag `<example>`, and ends with an end tag `</example>`. While most elements in a document are wrappers around content, empty elements are simply markers of occurrences. Empty elements have a modified syntax, where a trailing slash in an element tag, `<example/>`, indicates to a program processing the XML document that the element is empty and no matching end tag should be sought. Attributes are name/value pairs that occur inside tags after the element name. For example, `<Function name="do">` is the `Function` element with the attribute `name` having the value `"do."` In XML, all attribute values must be in quotes.

A Document Type Definition (DTD) contains a formal definition of a particular type of document; it is the definition of a schema, or information model, in XML form. This establishes what names can be used for elements, where they may occur, and how they all fit together. Thus, data can be checked for structural correctness (conformance to the DTD) and hierarchical data can be modeled to any level of complexity. An XML document that is syntactically correct is said to be *well-formed*; if a document also conforms to a declared DTD file, it is considered to be *valid*.

3.2 The Function Schema

The generic schema for the function information model is shown in Figure 1, where a word in brackets ("[ ]") indicates a reference to another data structure, and braces ("{ }") indicate a list of references to other data structures. The `Name` of the function is a string, and is required to be unique. The `Type` is a reference to a generic function class that is part of a function class taxonomy (to be discussed in Section 4). `Documentation` is a string used to describe the function. In cases where a description is somewhat long, this string can consist of or include file paths or Web universal resource locators (URLs) that lead to more information, images, etc. `Methods` is also a string and can also be a file path or Web URL. This item differs from `Documentation` in that `Methods` is intended to include computer-processable information (such as a computer program, code fragment, rules, constraints) to support computer-based reasoning about a design.

The next two items in the schema are `Input_flow` and `Output_flow`. These are references to lists of input and output flows for the function. The next item in the schema is `Subfunctions`. This item is a list of references to other function data structures, allowing a function to be decomposed into mul-

```
Function
    Name                string
    Type                [Generic_function_class]
    Documentation       string       (or NULL)
    Methods             string       (or NULL)
    Input_flow          {[Flow]}     (or NULL)
    Output_flow         {[Flow]}     (or NULL)
    Subfunctions        {[Function]} (or NULL)
    Subfunction_of      [Function]   (or NULL)
    Referring_artifact  [Artifact]
```

Figure 1. Generic Schema for Representation of Function

```
<!ELEMENT FunctionRepresentation (Artifacts, Functions, Flows)>

<!ELEMENT Functions (Function*)>
<!ELEMENT Function (Documentation?, Methods?, InputFlow,
    OutputFlow, Subfunctions?, ReferringArtifact)>
<!ATTLIST Function
    name ID    #REQUIRED
    type CDATA #REQUIRED
    subfunction_of IDREF #IMPLIED>

<!ELEMENT ref:Function EMPTY>
<!ATTLIST ref:Function
    ref IDREF #REQUIRED>
<!ELEMENT Subfunctions (Function | ref:Function)+>
<!ELEMENT ReferringArtifact (ref:Artifact)>
<!ELEMENT InputFlow (Flow | ref:Flow)*>
<!ELEMENT OutputFlow (Flow | ref:Flow)*>
<!ELEMENT Documentation (#PCDATA | Location)*>
<!ELEMENT Methods (#PCDATA | Location)*>
<!ELEMENT Property (#PCDATA)>
<!ELEMENT Location EMPTY>
<!ATTLIST Location
    type (path | URL) #REQUIRED
    value CDATA       #REQUIRED>

<!ELEMENT Artifacts (Artifact*)>
<!ELEMENT Artifact (#PCDATA)>
<!ATTLIST Artifact
     name ID #REQUIRED>
<!ELEMENT ref:Artifact EMPTY>
<!ATTLIST ref:Artifact
    ref IDREF #REQUIRED>
```

Figure 2. XML Mapping of the Generic Schema for Representation of Function

tiple subfunctions each of which may have its own associated input and output flows. As will be illustrated in an example below, the decomposition enabled by `Subfunctions` provides the means to map complex functionality to more detailed portions of an artifact model. The next item in the schema is `Subfunction_of`, which can be thought of as the inverse of a reference indicated by `Subfunctions`. In other words, if function A has functions B and C as subfunctions, then B and C are subfunctions of A and will list function A under `Subfunction_of`. The last item in the function schema is `Referring_artifact`. This is a reference from a function back to the artifact that references it.

The XML mapping of the generic function schema is shown in Figure 2; one can see a general mapping of items in the generic schema into the XML-based schema. A complete description of the XML specification is outside the scope of this paper and is not attempted here. Without delving into the details of XML, the following few syntactical points will aid those unfamiliar with XML in interpreting the schemata in this paper:

- Items in parentheses separated by a vertical line "|" indicate an either-or relation. Therefore, the definition of the `Subfunctions` element indicates that the data will either include a function, or a reference to a function.
- In general, a name that appears in the schema definition must have exactly one corresponding item in the data. For example, as can be seen in the third element definition in the schema, a `Function` must have exactly one referring artifact.
- Exceptions to the above rule are indicated by adding special characters to the end of a name. A name that ends in a question mark indicates an optional item that can occur zero or one times. Thus, the `Documentation` in the third element definition is not required, but if it occurs it can only occur once.
- A name that ends in an asterisk indicates an optional item that can occur zero or more times. Since the definition of `InputFlow` ends with an asterisk (combined with the either-or relation described above), it indicates that an In-

putFlow is a list of zero or more items, each of which can be either a flow or a reference to a flow.

- A name that ends in a plus sign "+" indicates a non-optional element that occurs one or more times. The definition of the Subfunctions element (which ends in a "+") indicates that the data will include a list of one or more items, each of which will be either a function or a reference to a function.
- CDATA and #PCDATA indicate generic data, i.e., strings. ID is also a string, but one that is required to be unique.
- Attributes are either required or optional, as indicated by the #REQUIRED and #IMPLIED keywords.

3.3 The Flow Schema

Figure 3 shows the schema for the flow information model. Like the function schema, the flow schema has a Name that is required to be unique, a Type (that references the generic flow class to which a given flow belongs, taken from a flow taxonomy to be discussed in Section 4), and a Documentation string. In addition to these items, the flow schema also has a Source and a Destination, which reference the physical artifacts that the flows for a given function enter from and exit to. These two items are shown with braces, indicating a list of references to allow the representation of a flow having multiple sources or destinations. Since a great many properties or parameters can be associated with flows, it would not be desirable to attempt to itemize *a priori* all the potential properties that could be relevant to a given flow and capture them in a monolithic data structure. The Properties item is a list of strings that specify flow properties or parameters, and can vary based on the user's requirements. For example, an electrical flow may have a property that specifies its voltage in a string such as "v = 5 Volts". The last item in the schema is Referring_functions, which is a list of references to the functions that have that particular flow as an input or output.

The mapping of the generic flow schema into XML is shown in Figure 4. Again, one does not need to be intimately familiar with XML to observe the general correspondence between information in the generic schema and the XML- based schema. Although the element that defines a flow includes Documentation as an optional element, the definition of the Documentation itself does not appear in Figure 4. The reason for this is that elements are only defined once. Since the Documentation element was defined in the function schema, it is not redefined as part of the flow definition.

3.4 Example

Figure 5 shows a schematic representation of the data structures and references for a mechanism that is part of a fluid pump design. This mechanism takes the rotational motion from a rotating shaft, which is driven by a motor, and converts it to oscillatory translational motion used to drive the pump piston heads. At one level, this mechanism can be considered as a single artifact, having an input flow (rotational motion) whose source is a motor shaft, and an output flow (oscillatory translational motion) whose destination is the pump heads. However, the function of this mechanism is actually more complex, consisting of multiple subfunctions each satisfied by different portions of the mechanism. The conversion of motion described above is accomplished as follows: a motor drives a shaft, which enters a gearbox; the gearbox reduces the speed of rotation, and the output motion drives a camshaft; the cam followers have links to the pump piston heads, resulting in an output at the piston heads that is an oscillatory translational motion.

```
Flow
    Name                string
    Type                [Generic_flow_class]
    Documentation       string       (or NULL)
    Source              {[Artifact]} (or NULL)
    Destination         {[Artifact]} (or NULL)
    Properties          {string}     (or NULL)
    Referring_functions {[Function]}
```

Figure 3. Generic Schema for Representation of Flow

```
<!ELEMENT Flows (Flow*)>
<!ELEMENT Flow (Documentation?, Source,
    Destination, Property*, ReferringFunctions)>
<!ATTLIST Flow
    name ID #REQUIRED
    type CDATA #REQUIRED>
<!ELEMENT ref:Flow EMPTY>
<!ATTLIST ref:Flow
    ref IDREF #REQUIRED>
<!ELEMENT ReferringFunctions (ref:Function)+>
<!ELEMENT Source (ref:Artifact*)>
<!ELEMENT Destination (ref:Artifact*)>
<!--COMMENT: The ELEMENT Property does not appear
here because it's defined in the function schema.
Each ELEMENT should only be defined once.-->
```

Figure 4. XML Mapping of the Generic Schema for Representation of Flow

These multiple subfunctions are represented individually and mapped appropriately back to the physical artifact domain, as shown in the figure. Although a discussion of a comprehensive design artifact representation is beyond the scope of this paper, representation of function is only one aspect of what is required for artifact modeling. Since the function and flow schemata include references to artifacts, boxes representing artifacts are shown in the figure. What is not captured at the function modeling level is the physical decomposition, i.e., the fact that this mechanism is a subassembly within the pump, and that the gearbox, cam and follower are parts of the mechanism assembly.

The XML-based representation of the pump mechanism function model is shown in Figure 6. Unlike the XML schemata defined previously, this figure illustrates the appearance of XML data, with references at the top indicating the version of XML being used, as well as the DTD file containing the Document Type Definitions for the schemata that the data must adhere to in order to be valid.

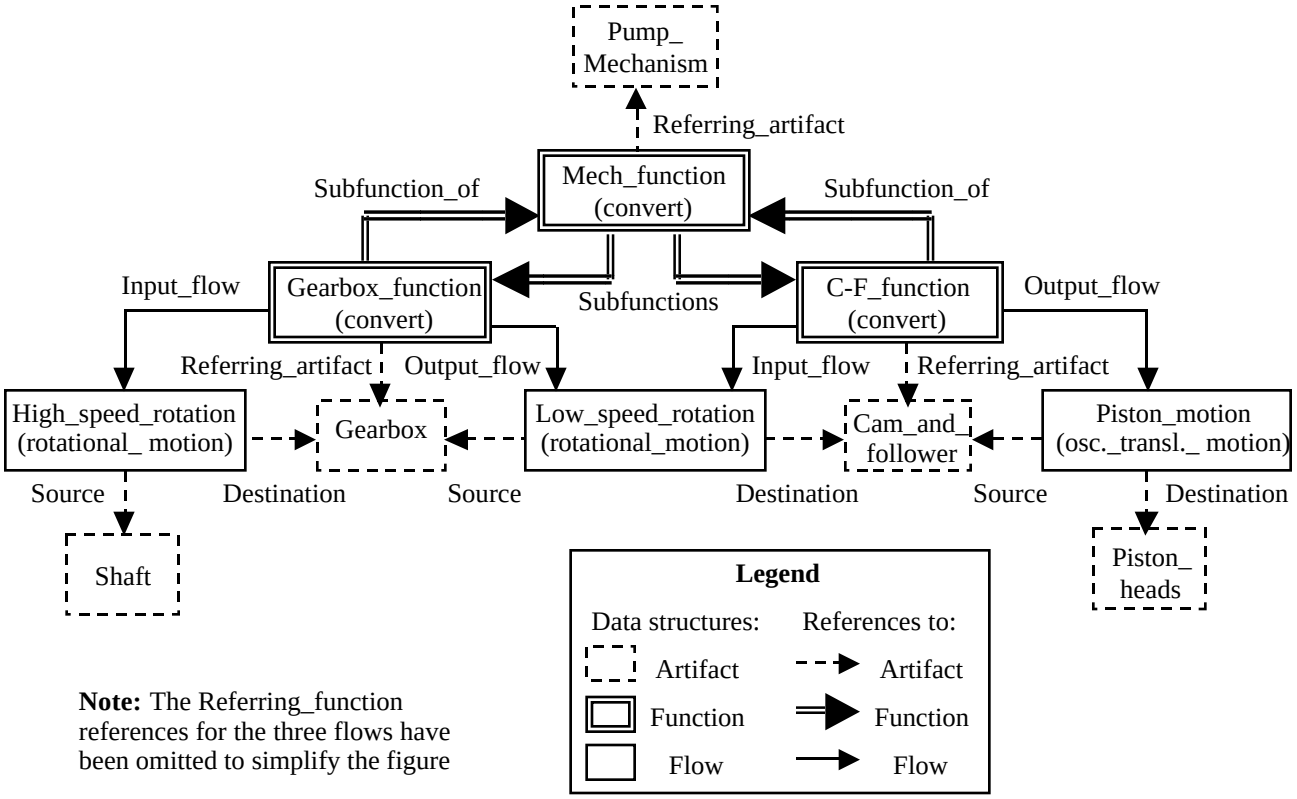


Figure 5. Graphical Illustration of Pump Mechanism Function Representation

Since an XML-based representation for artifact information has not yet been developed, the first set of definitions that appear are simply placeholders for artifact definitions. These are followed by the definitions of the three functions (the mechanism function and its two subfunctions), which include references to their associated input and output flows. These are then followed by the definitions of the three flows, which in turn provide the mapping of the function domain back to the physical domain via references to artifacts. These definitions are somewhat simplified for reasons of brevity, as this is merely an example. In an actual function model, additional information would be included in the form of documentation, lists of flow properties, and so on.

4 XML-BASED REPRESENTATION OF TAXONOMIES OF FUNCTION AND FLOW

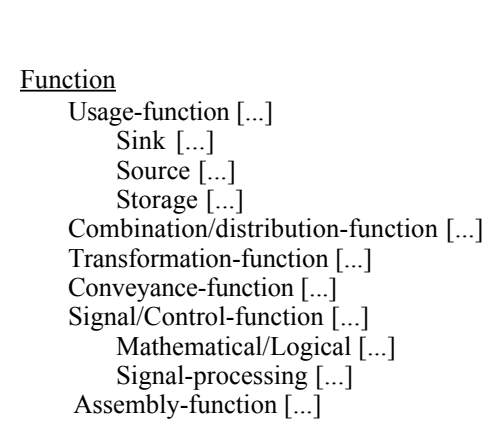
The function and flow schemata described in Section 3 both include a `Type`, which is used to reference the generic class to which that flow or function belongs. In addition to the development of these schemata, a second objective of this work is the development of generic taxonomies of function and flow that are concise, yet comprehensive enough to allow the modeling of a broad variety of engineering artifacts. The top-level divisions of the two taxonomies are shown in Figure 7. The indentation of terms identifies functions or flows that are subtypes of a more generic type; the bracketed ellipsis "[...]" indicate that each of the types listed actually has additional terms as subtypes that are not listed in the abbreviated taxonomies shown in the figure. The extended taxonomies of function and flow appear in (Szykman et al., 1999b). The taxonomies contain over 130 functions and over 100 flows. The evolution of both taxonomies to achieve more comprehensive coverage of engineering functions will be an ongoing part of this research.

In contrast to the schemata described previously, the generic schema for organizing the function and flow taxonomies is relatively simple. The organization of functions and flow is achieved through the definition of families. The generic schema for the family definition is shown in Figure 8. A generic function or flow type that has a set of subtypes (indicated by the indentation in Figure 7) defines a family with the subtypes as descendants. Aside from the highest level terms (the ones called "Function" and "Flow") any family must belong to another family, referred to as a `superFamily`. Families may also have documentation and properties, similar to those described for the schemata in Section 3.

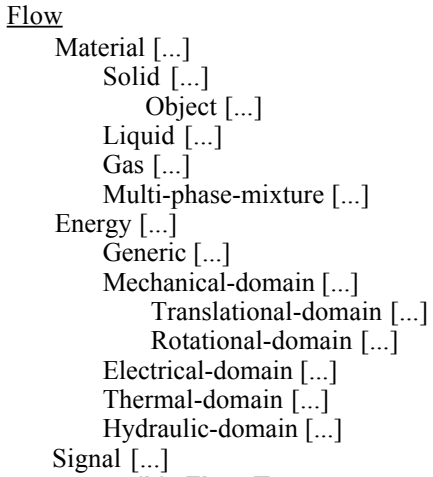
The mapping of the generic family schema into XML is shown in Figure 9. The `Name` of a family is a string and is required to be unique. The `superFamily` is a reference to the parent family. `Documentation` is a string used to describe the family. In cases where a description is somewhat long, this string can consist of or include file paths or Web URLs that lead to images or additional information. The `Properties` consist of a list of strings. These strings are used to represent information about the specific properties of a family such as the number of parameters necessary to describe one sort of family, inheritance constraints, etc. The last item in the schema is

```
<?XML version="1.0"?>
<!DOCTYPE Function-Representation SYSTEM "FunctionRepresentation.dtd">
<FunctionRepresentation>
<Artifacts>
  <Artifact name="Pump_mechanism"> ... </Artifact>
  <Artifact name="Shaft"> ... </Artifact>
  <Artifact name="Gearbox"> ... </Artifact>
  <Artifact name="Cam_and_follower"> ... </Artifact>
  <Artifact name="Piston_heads"> ... </Artifact>
</Artifacts>
<Functions>
  <Function name="Mech_function" type="convert">
    <InputFlow> </InputFlow>
    <OutputFlow> </OutputFlow>
    <Subfunctions>
      <ref:Function ref="Gearbox_function"/>
      <ref:Function ref="Cam_and_follower_function"/>
    </Subfunctions>
    <ReferringArtifact> <ref:Artifact ref="Pump_mechanism"/> </ReferringArtifact>
  </Function>
  <Function name="Gearbox_function" type="convert" subfunction_of="Mech_function">
    <InputFlow> <ref:Flow ref="High_speed_rotation"/> </InputFlow>
    <OutputFlow> <ref:Flow ref="Low_speed_rotation"/> </OutputFlow>
    <ReferringArtifact> <ref:Artifact ref="Gearbox"/> </ReferringArtifact>
  </Function>
  <Function name="Cam_and_follower_function" type="convert"
                          subfunction_of="Mech_function">
    <InputFlow> <ref:Flow ref="Low_speed_rotation"/> </InputFlow>
    <OutputFlow> <ref:Flow ref="Piston_motion"/> </OutputFlow>
    <ReferringArtifact> <ref:Artifact ref="Cam_and_follower"/> </ReferringArtifact>
  </Function>
</Functions>
<Flows>
  <Flow name="High_speed_motion" type="rotational_motion">
    <Source> <ref:Artifact ref="Shaft"/> </Source>
    <Destination> <ref:Artifact ref="Gearbox"/> </Destination>
    <ReferringFunctions> <ref:Function ref="Gearbox_function"/>
    </ReferringFunctions>
  </Flow>
  <Flow name="Low_speed_motion" type="rotational_motion">
    <Source> <ref:Artifact ref="Gearbox"/> </Source>
    <Destination> <ref:Artifact ref="Cam_and_follower"/> </Destination>
    <ReferringFunctions>
      <ref:Function ref="Gearbox_function"/>
      <ref:Function ref="Cam_and_follower_function"/>
    </ReferringFunctions>
  </Flow>
  <Flow name="Piston_motion" type="oscillating_translational_motion">
    <Source> <ref:Artifact ref="Cam_and_follower"/> </Source>
    <Destination> <ref:Artifact ref="Piston_heads"/> </Destination>
    <ReferringFunctions> <ref:Function ref="Cam_and_follower_function"/>
    </ReferringFunctions>
  </Flow>
</Flows>
</FunctionRepresentation>
```

Figure 6. XML-based Modeling of Motor Function Representation



(a) Function Taxonomy



(b) Flow Taxonomy

Figure 7. Top-level Subdivisions for the Function and Flow Taxonomies

```
Family
    Name          string
    superFamily   [Family]   (or NULL)
    Documentation string     (or NULL)
    Properties    {string}   (or NULL)
    Descendants   {[Family]} (or NULL)
```

Figure 8. Generic Schema for Organization of a Taxonomy

```
<!ELEMENT Taxonomy (Family+)>
<!ELEMENT Family (Documentation?, Property*,
    Descendants?)>
<!ATTLIST Family
    name ID #REQUIRED
    superFamily IDREF #IMPLIED>

<!ELEMENT Descendants (Family | ref:Family)*>
<!ELEMENT ref:Family EMPTY>
<!ATTLIST ref:Family
    ref IDREF #REQUIRED>
```

Figure 9. XML-based Modeling of the Generic Schema for Organization of a Taxonomy

`Descendants`. It is a reference to family entities that are specializations or subtypes of the current one.

To illustrate the use of the family schema to hierarchically organize the terms in a taxonomy, Figure 10 provides the definition of three of the families from the abbreviated flow taxonomy shown in Figure 7b: Flow, Material, and Solid. The generic term Flow is a family with three descendants, Material, Energy, and Signal. Material is a family whose `superFamily` is Flow, and which has four descendants. In the portion of the taxonomy shown in Figure 10, only the term Solid is broken down further; in the full taxonomy Material, Energy, and Signal, are all families having additional levels of descendants.

```
<?XML version="1.0"?>
<!DOCTYPE Taxonomy SYSTEM "Taxonomy.dtd">
<Taxonomy>
  <Family name="Flow">
    <Descendants>
      <ref:Family ref="Material"/>
      <ref:Family ref="Energy"/>
      <ref:Family ref="Signal"/>
    </Descendants>
  </Family>

  <Family name="Material" superFamily="Flow">
    <Descendants>
      <ref:Family ref="Solid"/>
      <ref:Family ref="Liquid"/>
      <ref:Family ref="Gas"/>
      <ref:Family ref="Multi-phase-mixture"/>
    </Descendants>
  </Family>

  <Family name="Solid" superFamily="Material">
    <Documentation>
      <Location type="URL"
          value="http://www.nist.gov/DRP/"/>
    </Documentation>
    <Property> property text A </Property>
    <Property> property text B </Property>
    <Descendants>
      <ref:Family ref="Object"/>
    </Descendants>
  </Family>
</Taxonomy>
```

Figure 10. XML-based Modeling of a Representative Portion of the Flow Taxonomy

5 APPROACHES TO INFORMATION PROCESSING, KNOWLEDGE RETRIEVAL, AND REUSE

Now that a formal specification for representation of artifact function information has been developed and mapped into XML to better support implementations using this representation, the next step is to begin generating algorithms and building software systems that use this representation. Any tools that will be used for processing information represented using these schemata and taxonomies will have to begin by reading in data from a file or database and parsing it—moving data from the file level into data structures at the software level. A significant advantage of the use of XML as an information modeling language is that generic XML parsers already exist, freeing developers from this burden.

Once XML data has been parsed, interfaces need to be developed to view and navigate this information. The next phase in this work is to develop an interface that will display and allow editing of the function and flow taxonomies. The development of other interfaces for more generic artifact modeling are discussed as part of the future work in the next section.

Some initial algorithms for information processing and reasoning about function have been identified and devised, but have not yet been implemented. One example is the method for identifying input and output flows for functions that are decomposed into subfunctions. Returning to the pump mechanism example discussed previously, one can see from Figure 5 that the mechanism function does not have input and output flows because it is decomposed into subfunctions. There is therefore a need for the ability to derive that information from the given data. This can be done as follows:

- Follow a decomposed function down the subfunction hierarchy to the lowest level where functions are not further decomposed, and create a list of all the functions (actually subfunctions) at the bottom-level. The decomposition only descends one level for the pump mechanism, but in the general case there may be multiple levels. The list of bottom-level level functions for the pump mechanism example consists of the gearbox function and the cam and follower function.
- Next, for each function in that list, add all the input flows to a list called *inputs* and add all the output flows to a list called *outputs*. In this example, the gearbox function will add high-speed rotation to the *inputs* list and a low-speed rotation to the *outputs* list. The cam and follower function adds low-speed rotation to the *inputs* list and oscillatory translational motion to the *outputs* list.
- Next, take any flow that appears in both lists, remove it from both the *inputs* and *outputs* lists, and add it to a list called *internal*.
- Any flows that remain on the *inputs* list are inputs to the decomposed function, and any flows remaining on the *outputs* list are outputs to the decomposed function. Any flows that end up on the *internal* list are "internal" to the higher-level decomposed function. In this example, the high-speed rotation is an input to the mechanism function, the oscillatory translational motion is an output. The low-speed rotation, which ends up on the *internal* list, is a flow that is internal to the pump mechanism but does not actually enter or leave the mechanism in that form.

Although it is not demonstrated by the simple pump mechanism example, this algorithm works with more complex functions that may be decomposed more than one level, and with subfunctions having multiple input and output flows that may reference different source and destination artifacts.

Once a large enough number of design artifacts has been modeled, the set of function models can serve as a database for knowledge retrieval and reuse. Using the above algorithm, queries can be made to search for solutions to particular function structures; a search for something that converts rotational motion to oscillatory translational motion, for instance, would turn up the pump mechanism. Since properties of flows are stored as lists of strings in the flow schema, these properties can be included as search criteria. A search can be made not only for a function structure that converts electrical energy to rotational motion, but one whose input is electrical energy in the form of direct current at no more than twelve Volts.

Furthermore, the formal representation that has been developed can support even more sophisticated types of queries. As illustrated by the graphical interpretation of the pump mechanism shown in Figure 5, the function representation can be mapped to a graph where the various data structures (functions, flows, artifacts) are nodes in the graph and references among data structures are arcs. Sophisticated graph-based pattern-matching algorithms have been developed in the field of computer science that could be used for even more intelligent kinds of searches. A search could therefore attempt to match not only a function with certain inputs and outputs, but could also target or avoid certain kinds of subfunctions or internal flows. And although a detailed representation of the artifact itself has not yet been developed, once such a representation has been implemented, searches could target or avoid certain kinds of artifacts, or could attempt to find function structures that satisfy certain functions and that have fewer than *X* parts, and so on.

Searches of these kinds are virtually impossible using traditional design databases where function is either described in some kind of natural language documentation, or even more often not captured explicitly at all. Thus, in addition to providing a neutral language for storing and exchanging function information, the representation described in this work provides a means for the creation of corporate design knowledge archives and repositories that support more effective retrieval and reuse of knowledge.

6 SUMMARY AND AREAS FOR FUTURE RESEARCH

This paper presents an XML-based mapping of a standardized representation of function consisting of schemata for functions and associated flows, along with taxonomies of generic functions and flows. The formal representation provides the means for representing functions that have multiple input and output flows, properties and parameters associated with flows, and the decomposition of functions into subfunctions each potentially having its own distinct flows. The representation also provides a mapping from the function domain to the physical domain (via references to artifacts in the flow schema) and supports the representation of function sharing provided that it is

used with an artifact representation that permits artifacts to have multiple functions.

Future work relating to the research presented in this paper is continuing along several avenues. To date, the focus of the development of the function taxonomy has focused on classes of functions that have flows associated with them. These are, however, not the only types of functions. One area of future work is to expand the scope of the taxonomy to cover different classes of functions, such as assembly functions and other non-flow-based functions relating to more abstract types of issues such as to *shelter* and to *provide access*. While the taxonomy does not currently extend to cover these other functions, the function representation itself is still capable of characterizing these concepts simply by using the function schema with no input and output flows.

The development of concise taxonomies containing generic terms has two main advantages. First, it reduces ambiguity at the modeling level, which can occur when multiple terms are used to mean the same things, or when the same term is used with multiple meanings. The distillation of a large body of terms into concise taxonomies does not eliminate this problem entirely, but it significantly lessens its occurrence. Second, it reduces the number of different was a given function concept can be represented. Both of these advantages provides benefits when it comes to indexing and retrieving function-based information from a knowledge base. However, because designers do use different terms to mean the same thing or use the same terms to mean different things, one potential contribution to this work would be to develop a "thesaurus" that would cross-reference multiple terms and multiple meanings, in order to aid a designer in finding the best term to use in a given context.

As engineering function is generally not of interest in the absence of any design artifact information, the representation of function proposed in this paper is intended to be incorporated as a layer within the context of a larger artifact modeling system that includes a more comprehensive representation of a design artifact. Other important aspect of design knowledge include geometry, behavior, physical (often hierarchical) decompositions, and other kinds of relationships. One such prototype system is being developed as part of the NIST Design Repository Project (Szykman et al., 1999a; Szykman et al., 1999c).

This prototype is now beginning a new implementation phase to produce a system architecture that utilizes XML-based representations of artifact information. This second-generation system will make use of the function representation presented in this paper, as well as representations that are currently being developed for other kinds of artifact knowledge. Existing interfaces for modeling artifacts and browsing artifact models will be re-implemented to interact with these XML-based artifact models. Part of this effort will include the modeling of various consumer products. This exercise will serve to validate the XML-based representations as well as the interfaces.

Beyond its adoption within the NIST Design Repository Project, it is hoped that the specification for a standardized representation of function presented in this paper will propagate into other parts of the academic and industrial research communities. In the near term, such a standardization will facilitate information exchange among researchers. In the longer term, this work is intended to provide a foundation for developers of the next generation of CAD systems and design tools. Aside from supporting ongoing research at NIST, helping to avert the undesirable emergence of multiple competing, proprietary formats for representing function—a problem that has adversely impacted industry in the area of geometric modeling and representation—has been a strong motivation for undertaking this work. Furthermore, should competing formats emerge, the proposed language may still be useful by providing a neutral exchange language much as STEP (ISO 10303, Standard for the Exchange of Product Model Data) (ISO, 1994a) is used for the exchange of geometric CAD data. Systems that have incompatible formats can still exchange information if both can read and write data in a standardized neutral format.

The specification for function representation provides a simple, generic language for representing function information. Consequently, the main contribution of this work is at the representational level. Algorithms for information processing and knowledge retrieval (including those described in the previous section) will have to be developed and implemented. This representation provides a generic infrastructure that will facilitate the capture and exchange of function information among researchers at present, and eventually in industry by contributing to interoperability between design systems, be they commercial or developed internally within a company.

Once a more complete artifact representation has been developed, another potential application for this work is to enhance the content of patent databases residing at the U.S. Patent and Trademark Office. Currently, patent information consists of text and two-dimensional images. A formal artifact representation that extends to function will provide a more comprehensive description of a device, while the combination of the formal representation and taxonomies of generic terms will allow more meaningful indexing, and will consequently enable more efficient search and retrieval of patent information.

REFERENCES

Baxter, J. E., N. P. Juster, and A. de Pennington (1994), "A Functional Data Model For Assemblies Used To Verify Product Design Specifications," *Proceedings of the IMechE, Part B, Journal of Engineering Manufacture*, **208**:235-244.

Chaudhri, V. K., A. Farquhar, R. Fikes, P. D. Karp, and J. P. Rice (1998), *Open Knowledge Base Connectivity 2.0*, KSL-98-06 (technical report), Stanford Knowledge Systems Laboratory, Stanford University, Stanford, CA.

Genesereth, M. R. and R. E. Fikes (1992), *Knowledge Interchange Format, Version 3.0 Reference Manual*, KSL-92-86 (technical report), Stanford Knowledge Systems Laboratory, Stanford University, Stanford, CA.

Hundal, M. S. (1990), "A Systematic Method for Developing Function Structures, Solutions and Concept Variants," *Mechanism and Machine Theory*, **25**(3):243-256.

ISO 10303-1:1994 (1994a), *Industrial Automation Systems and Integration – Product Data Representation and Exchange – Part 1: Overview and Fundamental Principles*.

ISO 10303-11:1994 (1994b), *Industrial Automation Systems and Integration – Product Data Representation and Exchange – Part 11: The EXPRESS Language Reference Manual*.

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

Lai, K., and W. R. D. Wilson (1989), "FDL - A Language for Function Description and Rationalization in Mechanical Design," *Journal of Mechanics, Transmissions, and Automation in Design*, **111**:117-123.

Lee, D. E. and H. T. Hahn (1998), "A Temporal Process Specification Language for Virtual Manufacturing Engineering," *Proceedings of the 1998 ASME Design Engineering Technical Conferences*, Paper No. DETC98/CIE-5535, Atlanta, GA, September.

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

Rodenacker, W. (1971) *Methodishes Konstruieren*, Springer, Berlin.

Shapiro, V. and H. Voelcker (1989), "On the Role of Geometry in Mechanical Design," *Research in Engineering Design*, **1**:69-73.

Stone, R. B. and K. L. Wood (1999), "Development of a Functional Basis for Design," *Proceedings of the 1999 ASME Design Engineering Technical Conferences (11th International Conference on Design Theory and Methodology)*, Paper No. DETC99/DTM-8765, Las Vegas, NV, September.

Sturges, R. H., K. O'Shaughnessy and M. I. Kilani (1996), "Computational Model for Conceptual Design Based on Extended Function Logic," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, **10**:255-274.

Szykman, S., J. W. Racz, C. Bochenek and R. D. Sriram (1999a), "A Web-based System for Design Artifact Modeling," *Design Studies* (accepted for publication).

Szykman, S., J. W. Racz, and R. D. Sriram (1999b), "The Representation of Function in Computer-based Design," *Proceedings of the 1999 ASME Design Engineering Technical Conferences (11th International Conference on Design Theory and Methodology)*, Paper No. DETC99/DTM-8742, Las Vegas, NV, September.

Szykman, S., R. D. Sriram, C. Bochenek and J. W. Racz (1999c), "The NIST Design Repository Project," *Advances in Soft Computing – Engineering Design and Manufacturing*, Roy, R., T. Furuhashi, and P. K. Chawdhry (Eds.), Springer-Verlag, London, pp 5-19.

Szykman, S., R. D. Sriram and S. J. Smith (Eds.) (1998), *Proceedings of the NIST Design Repository Workshop*, Gaithersburg, MD, November 1996.

Umeda, Y. and T. Tomiyama (1997), "Functional Reasoning in Design," *IEEE Expert Intelligent Systems and Their Applications*, **12**(2):42-48.

Veeramani, R., N. Viswanathan, and S. M. Joshi (1998), "Similarity-based Decision Support for Internet Enabled Supply-Web Interactions," *Proceedings of the 1998 ASME Design Engineering Technical Conferences*, Paper No. DETC98/CIE-5523, Atlanta, GA, September.

World Wide Web Consortium (1995), *Hypertext Markup Language - 2.0*, World Wide Web Consortium (W3C) Standard, September, <http://www.w3.org/MarkUp/html-spec/html-spec_toc.html>.

World Wide Web Consortium (1998), *Extensible Markup Language (XML) 1.0*, World Wide Web Consortium (W3C) Recommendation, February, <http://www.w3.org/TR/REC-xml>.