Proceedings of DETC'97
1997 ASME Design Engineering Technical Conferences
September 14–17, 1997, Sacramento, California

DETC97/DFM-4373

AN INFORMATION MODELING FRAMEWORK TO SUPPORT DESIGN DATABASES AND REPOSITORIES

J. W. Murdock[†]
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332

Simon Szykman[‡] **and Ram D. Sriram**
Manufacturing Systems Integration Division
National Institute of Standards and Technology
Building 304, Room 12
Gaithersburg, MD 20899

ABSTRACT

This paper introduces an information modeling framework to support representation of design artifacts for design databases and repositories. While most artifact representations consist primarily of geometric information, the object-oriented design modeling language developed through this work enables representation of not only form, but also function and behavior. This research has resulted in the implementation of a design artifact database as well as an information browser that provides the user interface to information contained therein. The implementation is demonstrated using the representation of a power drill as an example.

1 INTRODUCTION

This paper introduces an information modeling language that has been developed as part of the Design Repository Project at the National Institute of Standards and Technology (NIST). The long term objective of this project is the development of a framework to enable the creation of design repositories, that is, databases of design artifact and process information. Work in the area of artifact geometry has reached a level of maturity where international standards already exist for representation of geometric information. As a result of this maturity, this work makes use of STEP (ISO 10303) standards for representation of geometry. However, representation of non-geometric information is at a far more preliminary stage, and it is the development of a more comprehensive information model for design artifacts that is addressed in this paper.

Knowledge about the design of previous artifacts, and the process by which a design is realized are of great importance to designers. In order to create new and innovative designs, designers need an understanding of what has previously been designed and how those designs were realized. Merely providing access to schematics of artifacts (as done by the patent office) is inadequate for this purpose; this sort of information lacks the depth to convey a thorough understanding of an artifact. Even three-dimensional geometry serves a limited use in the absence of additional information. In contrast, detailed, coherent models describing not only geometry but also function and behavior are capable of providing enough information for truly effective design knowledge reuse.

Ultimately, the repositories created through the Design Repository Project will be used to collect and make available information about design artifacts and processes obtained through case studies developed at NIST, as well as contributed by industrial and academic partners. The intent is not to have a part catalog where a designer would search for a design to meet certain specifications. Rather, the repositories will act as a resource through which a designer can access case studies that are related to a problem of interest (similar artifacts, or similar design process), and abstract from them information that can be applied to a new problem.

The primary contribution of this work lies in the development of an object-oriented language for modeling design knowledge that is structured to aid in the objectives described above. The language is comprised of two components: the data lan-

[†] This research was performed while employed as a researcher at the National Institute of Standards and Technology.
[‡] Corresponding author.

guage and the design representation language. The data language describes the different types of data items in the information model (objects, relationships and their classes); the design representation language specifies the content of engineering artifact models implemented within this data language. This language is specifically suited to the problem of supporting clear and precise descriptions of an artifact for efficient and effective access and comprehension by a human expert designer. This work concerns not only information models which can potentially impact future (non-geometric) standards development, but also technology development through the creation of software tools for design data access and manipulation.

The remainder of the paper is organized as follows: Section 2 discusses related work and Section 3 introduces the modeling language, consisting of a data language and a design representation language; Section 4 describes the implementation of a design database that utilizes the modeling language to represent information, as well as the information browser that has been developed to serve as the user interface for accessing information stored in the database; Section 5 presents areas for future work.

2 RELATED WORK

Typical approaches toward product database management (PDM) and concurrent engineering support tools focus on database-related issues and do not place a primary emphasis on information models for artifact representation (e.g., Hardwick and Loffredo, 1995; Bliznakov et al., 1996; Kim, Han and Shin, 1996; Shah et al., 1996; Wood and Agogino, 1996). Although these systems often represent various types of information about the design process, representation of the artifact itself is generally limited to geometry.

Research in the area of intelligent design systems has taken an approach to artifact representation where modeling is typically divided into three main areas of study: the physical layout of the artifact (form), a causal account of the operation of the artifact (behavior), and an indication of the overall effect that the artifact creates (function). Examples of these areas include the qualitative simulation work in (de Kleer and Brown, 1983), behavioral and functional representation in (Iwasaki and Chandrasekaran, 1992), functional representation in (Chandrasekaran et al., 1993) and its successor "SBF models" from projects such as KRITIK (Goel, Bhatta and Stroulia, 1996) and INTERACTIVE KRITIK (Goel, Gomez et al., 1996), the YMIR project (Alberts and Dikker, 1994), CONGEN (Kim, Gorti et al., 1996), and others[1]. While there are major differences in the implementations of such models, the top level division into representation of form, behavior and function is a popular one.

This work utilizes a three-tiered approach to artifact representation which incorporates the three areas mentioned above, both because of the success of these areas in design computing and because this approach is inherently consistent with our own understanding of design. This research has resulted in an object-oriented representation format that provides a high level division into *form*, *behavior*, and *function*, which correspond to the three aforementioned areas of representation. The information associated with artifacts and with each of these three areas is described in detail in the following section.

[1] The FONM ontology (Hodges, 1995) adds a fourth area—the "intensional level"—which represents the use of a device in the broader context of an overall plan of action. This project does not make use of this level, but it may become appropriate to do so in the future.

3 THE MODELING LANGUAGE

The modeling language developed through this research consists of a data language and a design representation language. The data language provides a syntactic formalism for describing the way in which the information is represented. The design representation language represents the actual content of the artifact models. The device modeling language from the CONGEN architecture (Kim, Gorti et al., 1996) is used as a starting point for both of these interrelated aspects of the modeling language. CONGEN addresses the issue of partially-automated support for engineering design. However, its language was not a complete solution to the representation issues relating to the Design Repository Project; the needs of this project do not concern automated design support, but rather human-comprehensible representations. Thus the language from CONGEN has been adapted and extended for the purposes of this work. In general the modifications have been driven by two complementary factors:

- There are language features and properties that facilitate effective and efficient algorithms for reasoning about designs. These are necessarily present in CONGEN due to its intended role as an automated support tool, but are not required for this project. Along this dimension, some features of CONGEN have been simplified or eliminated for the language developed in this work.
- In contrast, this work presents a need for rapid and easy human comprehension of design data, which is less integral to the goals of CONGEN. This imposes a new set of constraints on the design of the modeling language. These constraints, in turn, drive the development of new language features and properties that are not present in CONGEN.

In general, the development of knowledge representations involves manipulating a trade-off between these two issues of computation and clarity. Because the Design Repository Project is not directly interested in pursuing the former, we claim that it makes a superior contribution to the latter.

3.1 The Data Language

Before proceeding to a description of the design representation language, it is necessary to briefly describe the representational formalism used to encode these results. The data language used in CONGEN is based on the SHARED object model (Wong and Sriram, 1993). That language provides capabilities similar to those in most object-oriented representational formalisms but is structured to make formal reasoning and analysis easier than for other object models.

This research has developed a database for storing data using the modeling language, as well as tools for examining and manipulating this data. It is expected that such databases will eventually be used on a large scale in a distributed environment (i.e., over the Internet). Thus there is a requirement that the data language enable effective and efficient solutions to issues such as storage, indexing, concurrent access, etc. These sorts of problems are fundamentally different from those involved in automated design reasoning performed in CONGEN. Furthermore, these tasks do not require the same level of mathematical rigor required to achieve partial-automation of design tasks.

Therefore, this work proposes a data language which is superficially similar to SHARED, but which is substantially clearer and more concise at the cost of being less rigorous. The syntactic mechanisms used in this project are approximately, but not exactly, a subset of the SHARED model. However the

Object: Robert_Smith	
parent	Employee
uid	12
Attributes	
title	"Junior Programmer"
address	[address_1]
salary	27520.00
Relationships	
[has_supervisor_15]	
[in_office_13]	

Figure 1: Example of an object.

Class: Employee	
parent	Person
cid	26
Attributes	
title	STRING
address	[address_class]
salary	FLOAT
Relationships	
[has_supervisor_rclass]	
[in_office_rclass]	

Figure 2: Example of a class.

precise semantics associated with those features that have been retained are, in some instances, significantly different from the analogous constructs within SHARED. Thus, the data language can be more accurately characterized as a separate, but similar, language rather than as a restricted subset.

Within the data language, there are four basic types of data items: objects, classes, relationships, and relationship classes[2]. At this level, the object model is generic enough that it could be used for many applications other than representation of engineering data. The design representation language and the engineering context are not discussed until Section 3.2. Thus, to illustrate the basic building blocks of the data language independently of engineering-related issues, Section 3.1 uses examples from outside the domain of engineering design. An engineering artifact (a Black & Decker® power drill) is used for the implementation example described in Section 4.

3.1.1 Objects. Figure 1 contains an example of an object (data items are denoted by a data item name in brackets). This object defines a person named Robert Smith as might appear in a personnel database. In this example, Robert Smith is given a title (a text string), a salary (a number), and an address; the address is not represented by a string but is contained in a separate object. Furthermore, Smith is involved in two explicit data relations (represented by relationship data items): a relationship which identifies his supervisor and one that identifies the office in which he works.

In general, an object contains the following information: a name, a parent[3], a unique identifier (*uid*), a set of attribute-value pairs, and a set of relationships in which it is involved. The parent indicates the class from which the object is instantiated. Internally to the database, data items are referenced using unique identifiers (which are integers); externally, they are presented in the user interface by name, as illustrated by the objects containing Smith's address, supervisor and office in Figure 1. The attribute-value pairs each have an attribute name and a value which can be a object, a set of objects, or a primitive data item such as an integer, a string, or a floating point number. The relationships portion of the object lists all relationship data items for which the object is a participant.

The data language is silent as to which connections between objects are represented using attributes and which are represented using relationships. For example, the **Robert_Smith** object could have just as easily had a **supervisor** attribute and a **has_address** relationship instead of the other way around. The decisions about which connections are defined explicitly using relationships and which ones are inherent to the nature of the particular objects being described is part of the content theory of the domain being described. Thus in this project, decisions about what is an attribute and what is a relationship are part of the design representation language described in Section 3.2.

3.1.2 Classes. Figure 2 shows an example of a class. The Employee class is the class of which the Robert_Smith object is a member and is a subclass of the general Person class. The Employee class defines attributes and types for objects of that class, and refers to the relationship classes for the relationships which generally relate to Employee objects.

Every class has a name, a unique class identifier (*cid*), a set of attribute-type pairs which define attributes held by members of the class along with their corresponding classes or primitive data types, and a set of relationship classes that define the relationships in which objects of that class participate. Furthermore, a class may have a parent of which that class is a subclass; top level classes have no parent.

In general, the attributes and relationship classes of a class will be a superset of those of its parents. For example, the **Person** class would have an address attribute but no title or salary attributes. If there were a **Customer** class (also subclass of **Person**), it would have the same address attribute from the **Person** class but might also have additional attributes and relationships relating to orders placed, bills outstanding, etc. In addition to adding new slots, particular slots from a parent class may remain unused if they are not relevant to that particular subclass. Note that an object belonging to a class will generally have the slots defined by the class that it is a member of, but may also choose to add extra slots and/or not use unnecessary ones. Thus class definitions describe the kind of content that is normally present in them, but do not require the content to be fully specified. This is consistent with the fact that this data is intended to be used strictly as a presentation to the user and not as a foundation for a programming environment.

[2] Although the four data items are all "objects" in the object-oriented sense, in this paper the term "object" is not used in a generic sense but rather denotes a *specific* type of data. When describing *generic* data structures, the term "data item" will be used.

[3] Also referred to as an *instance-of* or *is-a* link in the database literature.

Relationship: Has_supervisor_15	
parent	Has_supervisor
rid	17
Roles	
supervisor	[Michael_Thompson]
subordinates	{[Robert_Smith], [Jack_Evans], [Tom_Williams], [Steve_Jones]}
Attributes	
group_budget	250000.00

Figure 3: Example of a relationship.

Relationship Class: Has_supervisor	
parent	
rcid	6
Roles	
supervisor	[employee_class]
subordinates	{[employee_class]}
Attributes	
group_budget	FLOAT

Figure 4: Example of a relationship class.

3.1.3 Relationships. Figure 3 provides an example of a relationship. In this relationship, Smith is described as being one of the employees who's supervisor is the employee Michael_Thompson. The relationship has two roles: the supervisor which refers to a single employee and the subordinates which refers to a set of employees (denoted by braces). In addition the relationship defines the budget for the entire group as an attribute of the relationship between these employees.

In general, a relationship has a name, a unique relationship identifier (*rid*), a set of roles and values for those roles (which must be objects that have the relationship in their list of relationships), and a set of attribute-value pairs. The relationship in Figure 3 is a binary one-to-many relationship (i.e., there is one supervisor, and a set of subordinate employees each of which have this relationship in their object). In general, the data language supports any number of roles, any of which may involve either single values or sets of values. Thus one-to-one, many-to-many, one-to-many-to-many relationships, etc. are supported.

3.1.4 Relationship Classes. Figure 4 shows an example of a relationship class that provides a general definition of the relationship between a supervisor and a set of subordinates, annotated with a budget. This relationship class is a top level class and thus has no parent; in general, however, there may be superclasses and subclasses for relationship classes just as there are for (object) classes.

Each relationship has a name, a unique relationship class identifier (*rcid*), a set of role-type pairs whose types are defined as either containing an object of a specific class (as in the supervisor role in the figure) or a set of objects of a specific class (as in the subordinates role in the figure), and a set of attributes which are defined exactly like attributes for a class. As with the classes, subclasses and instances may add or leave unused roles and/or attributes from a relationship class as needed.

3.2 The Design Representation Language

Section 3.1 introduced the generic components of the data language. This section presents the design representation language, which uses the structure of the data language to represent engineering designs. Sections 3.2.1 and 3.2.2 briefly describe the elements of knowledge that are represented using this language; Section 3.2.3 discusses further the engineering context associated with the representation.

3.2.1 Artifacts. The central constituent of the design representation language is the artifact. Every physical object described in this language (such as the power drill that will be presented in Section 4) is represented as an artifact. A critical aspect of the design representation language is the fact that the subassemblies and components of an artifact are themselves artifacts. For example, the motor, which is an artifact, is part of the drill system, which is also an artifact, which is part of the overall power drill assembly, also an artifact. Thus the language is truly hierarchical: elements at different levels of the hierarchy have the same basic content (i.e., form, behavior, and function) and representational structure. The representation of a device design is comprised not only of the collection of artifact objects, but also the other objects, relationships, the connections between data items, as well as various attributes and their values.

Using the design representation language, any artifact is an instance of some type of artifact class. For example, the power drill artifact is an instance of the **Drill_artifact** class which is a subset of the general **Artifact** class. The general **Artifact** class provides two attribute slots for annotating the artifact, both of type string: a **full_name** slot for providing a complete technical name and a **description** slot for providing a brief textual description. The **Artifact** class also includes attributes that provide the top level division of the artifact into the three basic elements of form, behavior, and function, and therefore artifacts instantiated from subclasses of the general class can contain these elements. Each of the three elements contains a reference to another object that describes that aspect of the artifact further through references to additional objects.

3.2.2 Form, Behavior, and Function. Of the three elements of representation, the primary focus of this research is on function. Much of the work required in the areas of functional and behavioral modeling is due to a lack of established standards for representation of such information. In contrast, representation of form can be achieved using existing standards, specifically STEP AP 203. Although this standard is not "complete," meaning that certain types of CAD information are not captured by STEP AP 203, STEP compliance is becoming a part of more and more CAD tools. To enable the most widespread compatibility with existing tools and to avoid duplication of substantial effort, form in this work utilizes STEP AP 203, thereby eliminating the need for development of a novel representation to encode form.

In this project, work to date for representation of behavior includes a simplified characterization of behavior that allows the user to specify a set of one or more inputs and outputs, as well as relationships, such as being a subbehavior of a composite behavior (i.e., a behavior that is satisfied by multiple objects). The representation scheme for this element will be developed further as subsequent research reveals additional needs.

The long-term goal of this research is to enable large scale design databases and design repositories to be used to help learn about requirements for design and redesign of a given type of artifact. Designers who access a design repository will generally be searching for designs to perform a given function (often without specifying how the function is achieved) rather than searching for designs that behave a certain way. Thus, for the purposes of this work, function is of significantly greater importance than behavior.

The language for describing function within the overall design representation language is significantly more mature. Functions are instantiations of classes of functions. The function of the motor, for example, is an instance of the **Transform_function** class which is, in turn, a subclass of the general **Function** class. A function has a set of inputs and outputs, and may have other attributes or relationships that specify subfunctions and composite functions.

A function in the design representation language describes an interaction between *fluents*. More specifically, inputs and outputs of functions are described in terms of sets of instances of a subclass of the general **Fluent** class. The term *fluent* comes from (Russel and Norvig, 1995), which uses it in a slightly broader sense than this research does. In the context of this work, fluents include both physical and abstract phenomena that are associated with inputs and outputs to functions, such as motion, force, heat, liquids, current, and so on. The usage of fluent in this paper is essentially the same as the notion of a *substance* as used in design computing projects such as (Bylander, 1991), (Chandrasekaran et al., 1993), and (Goel, Bhatta and Stroulia, 1996). Use of the term *substance* in this work has been intentionally avoided because of the common confusion it creates with the non-technical definition of *substance* which refers only to physical substances and would exclude fluents such as motion or force.

The fluent classes all have slots for explicit references to which artifact the fluent is flowing from and where it is flowing to, as well as a slot for relevant parameters. It should be noted that fluents are not a type of top-level design artifact information as are form, behavior and function. Rather, the set of subclasses for the general **Function** and **Fluent** classes together describe the semantics of the functional aspects of this design modeling language. This concept will be discussed further in the following section.

3.2.3 The Engineering Context. Section 3.1 described the basic building blocks of the data language, which forms the framework around which the design representation language is structured. As the personnel database example illustrated, the data language is quite generic. What distinguishes the design representation language from the simple nature of the data language (and from other generic object-oriented approaches to information modeling) is the engineering context that is built into the language. Although a comprehensive description of the engineering context is beyond the scope of this paper, this section is intended to provide some insight to the reader since it is this context that gears the design representation language specifically toward the representation of engineering design knowledge.

The engineering context has several different aspects. These include taxonomies of functions and fluents (both of which include a class hierarchy and instances of those classes) as well as the types of attributes and attribute values associated with the objects in a design representation[4]. The function hierarchy consists of four subclasses of function directly beneath the main **Function** class. These are **Transform_function**, **Convey_function**, **Supply_function** and **Control_function**, which are defined as follows:

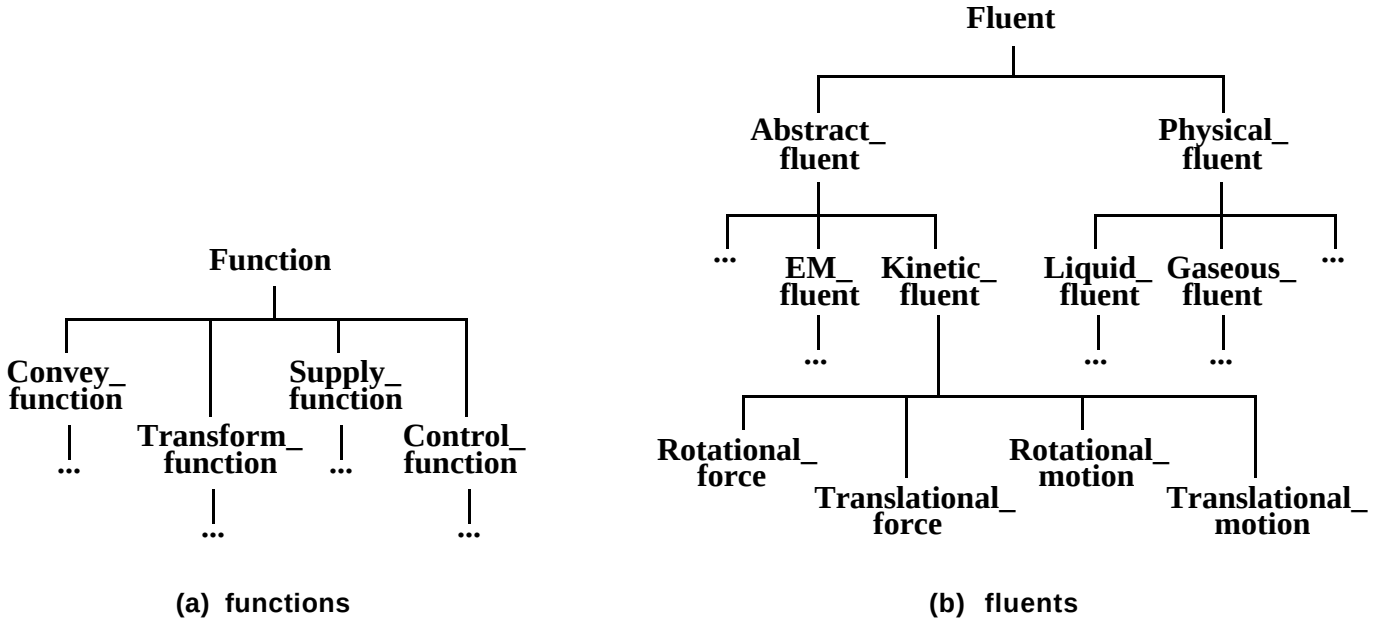
- **Convey_function**: The transfer of a fluent from one location to another. For example, the function of a wire may be to convey current from a power source to a motor.
- **Transform_function**: The conversion of one fluent into another. For instance, the function of a motor may be to transform electricity into rotational motion.
- **Supply_function**: The production of a fluent. For example, the function of a power source might be to supply electricity to a motor.
- **Control_function**: The exertion of a set of effects on fluents in the system. For instance, the function of a switch may be to enable rotational force from a motor by closing a circuit. There is a separate hierarchy of classes of effects, such as actuation, inhibition, enabling, adjustment of parameters, etc.

The top-level division within the **Fluent** class is into the two main subclasses: **Abstract_fluent** and **Physical_fluent** (Goel, Bhatta and Stroulia, 1996). The abstract fluent classes specify things like motion and electricity while the physical fluent classes specify things like gases and liquids. To illustrate, the function of the drill motor is to convert electrical energy to rotational motion. Electricity coming into the motor is of the **Direct_current** class which is a subclass of the **EM_Fluent** (short for electricity and magnetism) class, which is in turn a subclass of **Abstract_fluent**, which is in turn a subclass of **Fluent**.

These fluents may then be further annotated by parameters which specify particular quantitative or qualitative traits that a fluent might have. For example, the output of the motor is rotational motion; the speed and direction of this motion may vary depending on the settings of the trigger switch and control lever and thus are explicitly specified as parameters of this rotational motion which may be affected by those characteristics. Figures 5(a) and 5(b) illustrate abridged portions of the function and fluent taxonomies. Currently, all functions are instances of one of the four main classes shown in Figure 5(a). The incorporation of more detailed function taxonomies (e.g., Little et al., 1997) is an area of future work.

Another essential part of the engineering context is supplied by relationships. Relationships provide the means for creating a physical or functional decomposition for an artifact by allowing the representation of subassemblies and subfunctions. Relationships are also uses to describe control structures by specifying how certain effects are determined by values of various parameters. In this manner, the effect of the trigger switch on the drill bit speed can be represented.

[4] Equally important is the hierarchy of artifact objects. However this knowledge is not part of the general engineering context because this hierarchy—in essence the decomposition of an artifact into assemblies, subassemblies and components—is specific to the design being represented.



(a) functions **(b) fluents**

Figure 5: The function and fluent taxonomies.

4 IMPLEMENTATION

4.1 The Tool Suite

The implementation of the research described in this paper is a tool suite consisting of several different components. The tool suite itself uses an object-oriented database management system called ObjectStore™, developed by Object Design, Inc.[5] Non-geometric information about a design artifact (e.g., function, behavior, relationships, etc.) is initially stored in a plain text file which follows a prescribed format. This information is then transferred to the object-oriented database through the use of a design data compiler which acts as an input interface to allow the transfer of information to the database. A similar application is available to extract the contents of the database and save them as a single flat file in a predetermined format.

Artifact information stored in the database is accessed through an information browser (not to be confused with a web browser) that provides the user with a point-and-click interface that is easy to use. The information browser (illustrated in the next section) allows the user to explore the artifact information by moving from a given data item to others that are related through behavior, form, function and relationships. A command-line interface to the same functionality provided by the browser can also be used when the use of a graphical interface is either not desired or not possible.

Within the database, objects corresponding to assemblies, subassemblies and components contain links to the corresponding geometric data in STEP format. By clicking on a button, the user can view the geometric model of an object of interest. The visualization capability utilizes STEP/Works, a 3D visualization tool for AP 203 geometries developed by International TechneGroup, Inc. Aside from ObjectStore™ and STEP/Works which are commercial products, the tool suite is implemented using a variety of languages including C++, Flex, Andrew Bison, and Tcl/Tk, and runs on the Unix platform.

[5] The identification this and other software and hardware products in this paper is intended to provide readers with information regarding the implementation of the research described; no approval or endorsement of any product by the National Institute of Standards and Technology is intended or implied.

4.2 Example

This section describes the implementation of an artifact database utilizing the modeling language presented in this paper. Figure 6 illustrates the Black & Decker® VP840 power drill that is used as an example to illustrate the design modeling language. The object corresponding to the drill, **drill_artifact_1**, is depicted in Figure 7, which shows the information browser. Information for some of the fields consists of text, while other fields contain buttons; clicking on any of these buttons will bring up the data item associated with that field. Since **drill_artifact_1** is an instance of the **Drill_artifact** class, clicking on the parent button brings up the **Drill_artifact** class data item, which in turn has the general **Artifact** class as a parent.

Clicking on the **has_part_1** button brings up the relationship shown in Figure 8, which is an instance of the **Has_part** relationship class. This relationship describes the composition of components into the artifact as a whole, dividing the drill into a set of components and subassemblies such as the drill system or some of the wires that connect the systems together. Note that there is no explicit distinction made between objects which are basic components (e.g., **white_wire_1**) and those which are themselves complex systems (e.g., **drill_system_1**); the only difference in the database is that complex systems also have additional **Has_part** relationships describing their decomposition into subsystems and components.

Clicking on any of the component buttons will bring up new objects, some of which may be further decomposed, such as the drill system consisting of a motor, clutch, chuck, etc.

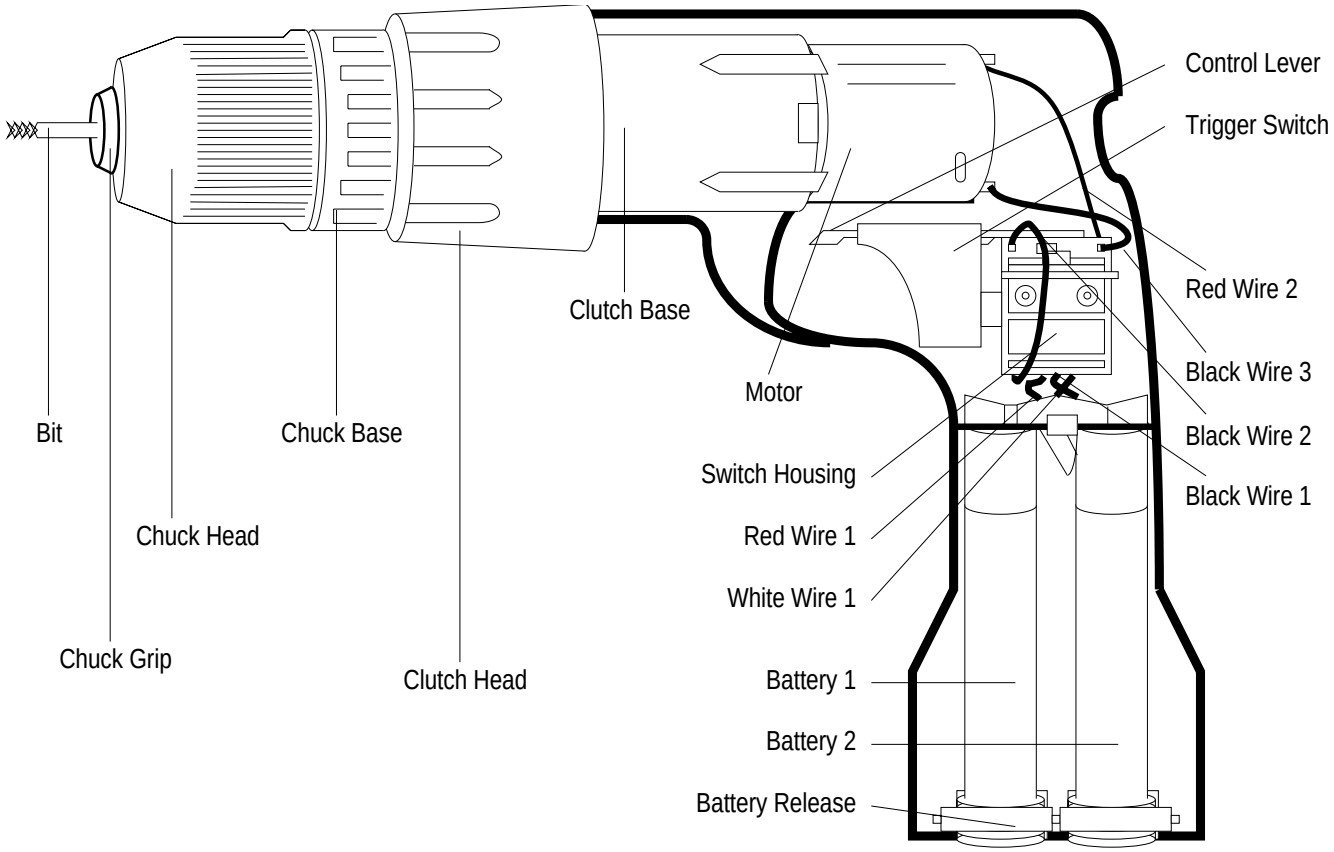


Figure 6: The Black & Decker® VP840 power drill.

The modeling language does not necessarily require that the primitive elements be at any specific level of detail. For example, several components, such as the motor, are considered to be OEM (original equipment manufacturer) catalog parts. These are not decomposed further since detailed artifact information may not be available or necessary. If they were designed in-house, more detailed representations of the subcomponents which form the motor would be included. Clicking on a button for form brings up a data item associated with the artifact's form, and a button there allows the user to view the geometric model of the artifact by calling the STEP/Works viewer.

The object for the motor, although not shown, looks similar to the object shown in Figure 7. Clicking on the function button brings up the function shown in Figure 9. This object is an instance of the **Transform_function** class, and has two attributes that correspond to the fluents on which this function acts. The objects corresponding to those fluents are shown in Figures 10 and 11. The three objects shown in Figures 9 – 11 taken together indicate that the function of the motor is to transform direct current, which enters the motor from a black wire, to rotational motion, which goes from the motor to the base of the clutch. It can also be seen that the motor function output (Figure 11) has certain parameters associated with it regarding the speed and direction of the rotational motion. These parameters relate back to the control system (see Figure 8) since the speed is controlled by the trigger switch and the direction by the control lever.

Using the browser, the user can view data items associated with those parameters and could, with a few clicks, reach the representation of the trigger switch assembly or return to the relationship shown in Figure 8 via links through the control system instead of the drill system. The browser itself provides an intuitive interface that allows the user to easily browse a database and rapidly extract information about the various aspects of the artifact representation. The overall drill artifact database contains 28 artifact objects (assemblies, subassemblies and components) and over 250 additional object and relationship data items belonging to 64 (object) classes and 6 relationship classes that describe their form, functions and behaviors. The data items shown in Figures 7 – 11 illustrate only a few of the many types of attributes present in the database. Other attributes include color, material, control function settings (e.g., a switch position), and other parameters such as the battery voltage, output torque, and direction of rotation.

5 AREAS FOR FUTURE RESEARCH

This paper has presented an information modeling framework to support the creation of design artifact databases and repositories. This modeling language consists of a data language and a design representation language, which together enable the representation of form, behavior and function, in contrast to traditional representations which focus primarily on geometry.

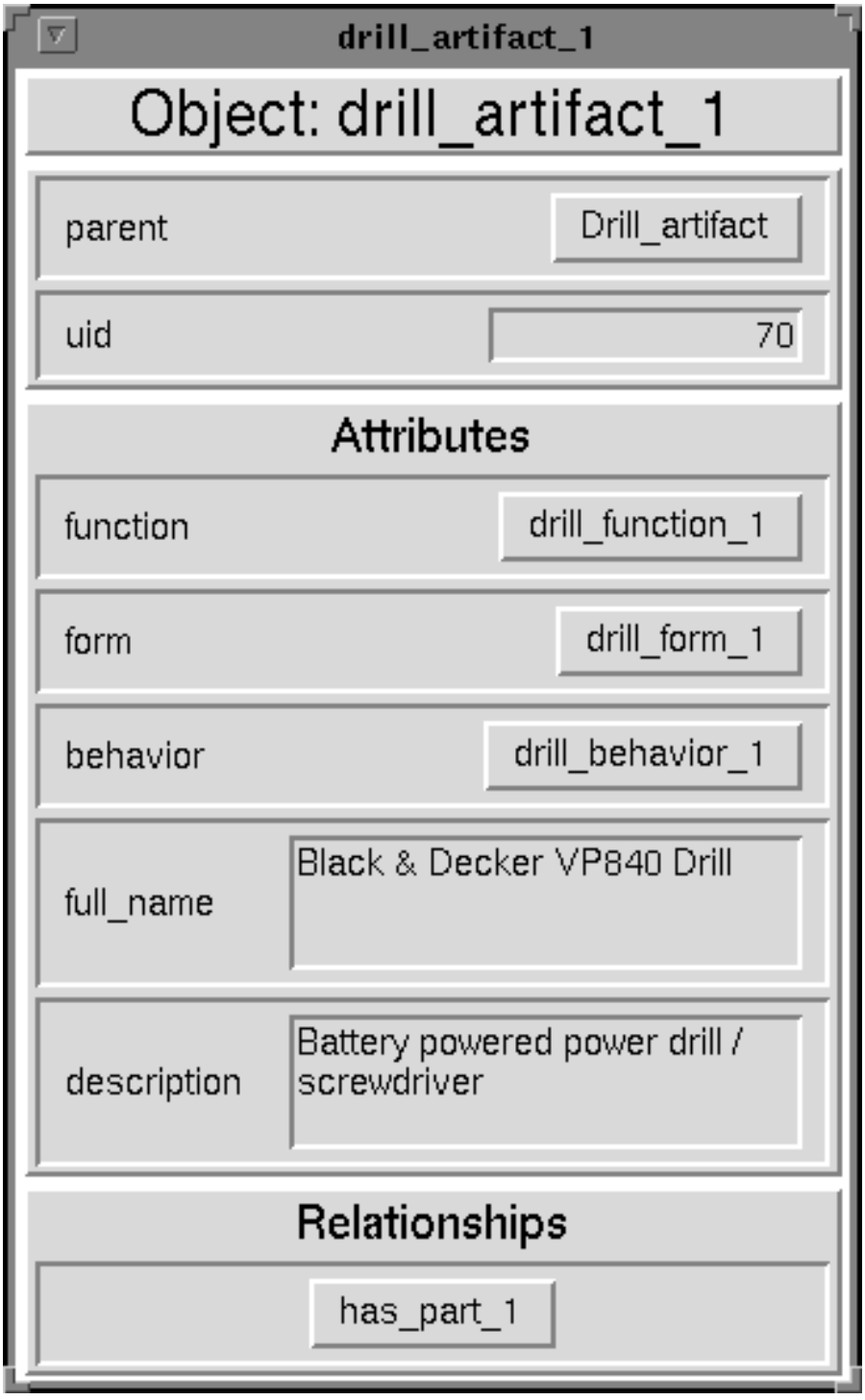


Figure 7: The drill artifact object.

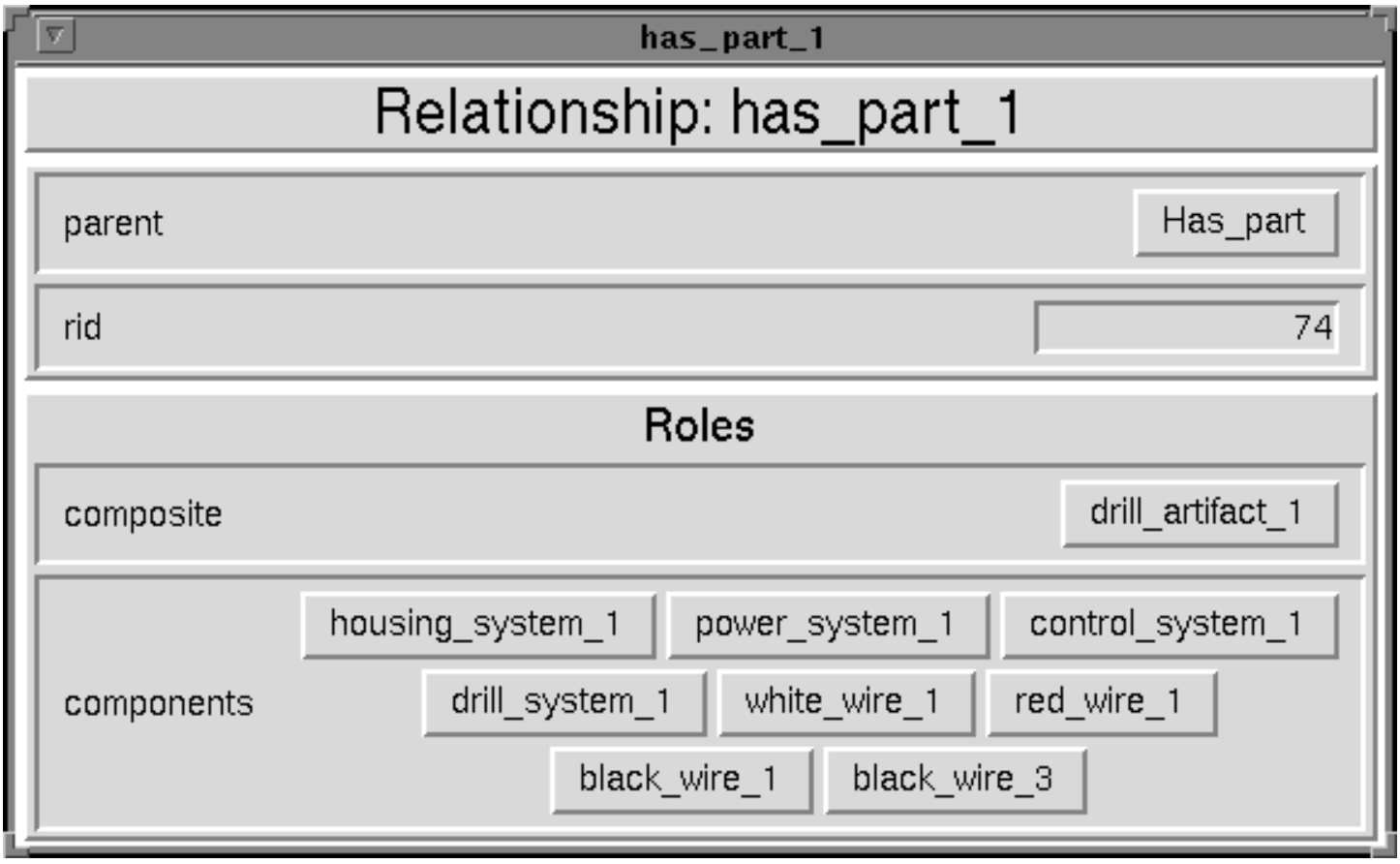


Figure 8: The relationship representing the top-level decomposition into systems and components.

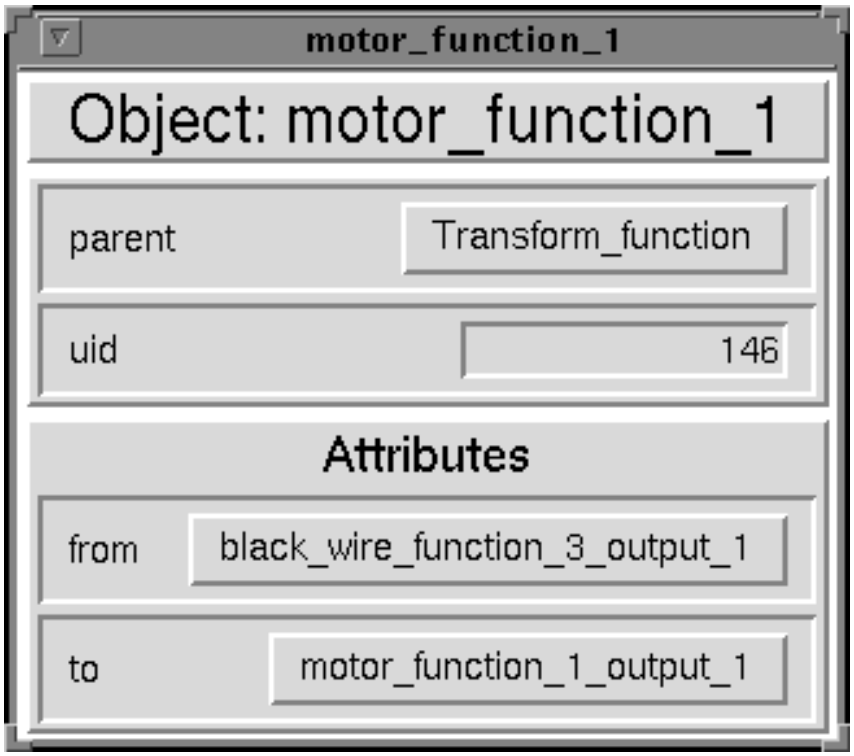


Figure 9: The object for the drill motor function.

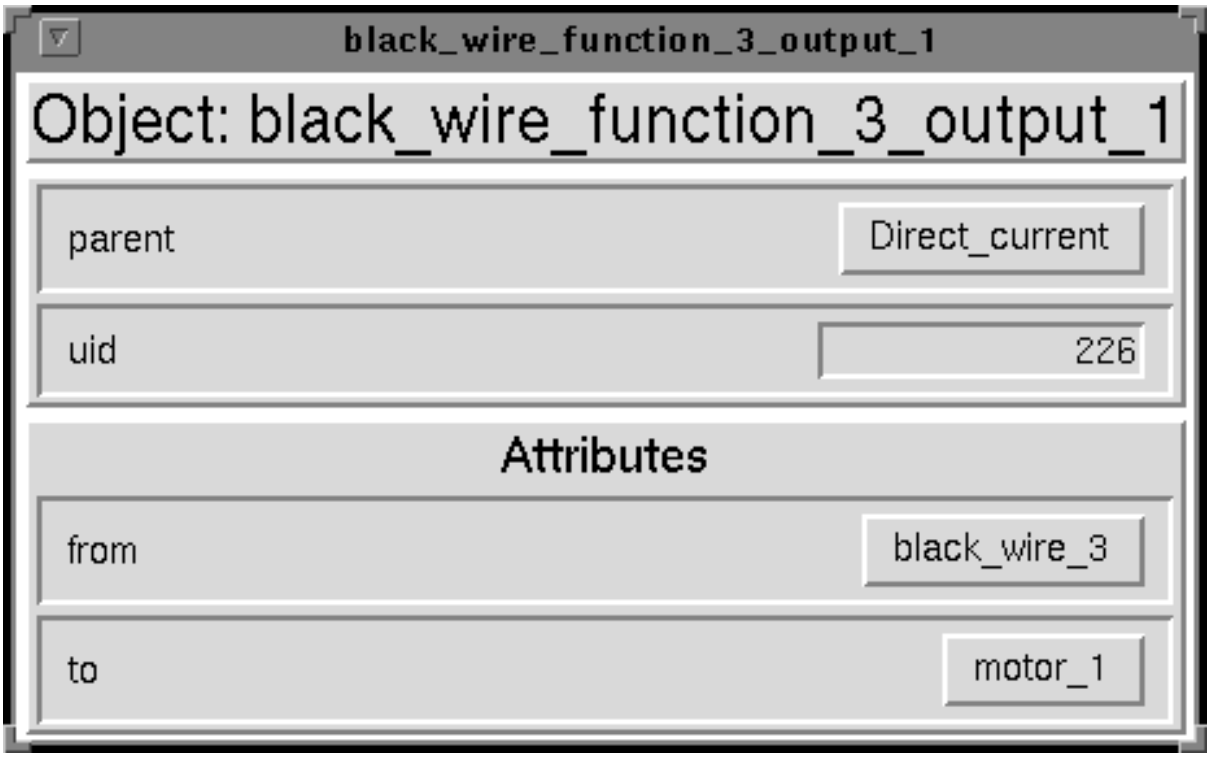


Figure 10: The object corresponding to the input fluent for the drill motor function (note: wire output is equivalent to motor input).

There are several areas of ongoing and long-term future work associated with this work. Current research is focusing on developing a web-based interface to the engineering artifact databases. This interface will be similar to the browser illustrated in the previous section, but rather than being stand-alone application, information will be delivered to the user via the web using a web browser. Because web browsers exist for many platforms, using a web browser as an interface makes access to the databases platform-independent, obviating the need for recreating our browser for multiple platforms. Another advantage of this approach is the relative ease with which information contained in multiple (possibly distributed) databases can be delivered to users who may also be geographically distributed.

As stated in Section 3, the current representation of behavior is somewhat simplistic. One of the primary objectives for future work on this project is to develop a clear and powerful account of artifact behavior which is consistent with both the representation of function that has been developed and the existing STEP standard that is used to represent its form. Because the behavior of an artifact is defined to be the way in which its form accomplishes its function, the representation of behavior is clearly heavily dependent on both the representations of form and function.

The current design modeling language is highly flexible in its ability to represent knowledge about a design. However, in order for a design repository to be useful, there is a need to impose structure onto the data. This is particularly important for the indexing problem, where terminology issues (such as terms with multiple meanings, or multiple terms with one meaning) can serve as barriers to finding data even when it exists in the database. Long term work will require the investigation and development of ontologies (generic base classes, as well as domain-specific classes) (see Farquhar et al., 1995).

ACKNOWLEDGMENTS

This work was supported in part by the National Research Council Postdoctoral Research Associateship Program. Additional support was provided by the NIST SIMA program and the DARPA RaDEO program.

REFERENCES

Alberts, L. K. and F. Dikker (1994), "Integrating Standards and Synthesis Knowledge Using the YMIR Ontology," *Artificial Intelligence in Design '94*, J. S. Gero (ed.), Kluwer Academic Publishers, Boston.

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

Bylander, T. (1991), "A Theory of Consolidation for Reasoning about Devices," *Man-Machine Studies*, **35**:467-489.

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

de Kleer, J. and J. S. Brown (1983), "Assumptions and Ambiguities in Mechanistic Mental Models," *Mental Models*, D. Gentner and A. L. Stevens (eds.), Lawrence Erlbaum Associates.

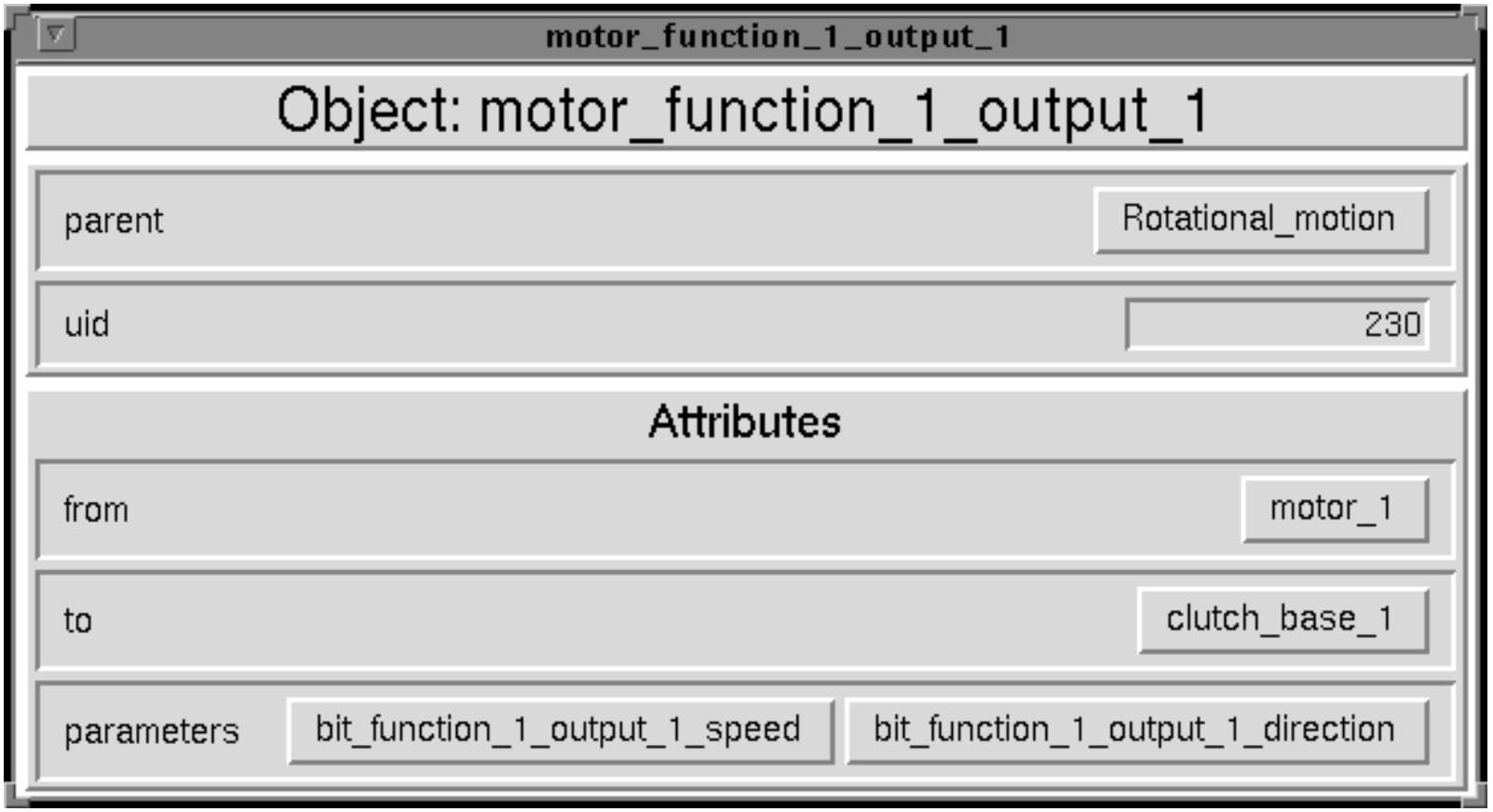


Figure 11: The object corresponding to the output fluent for the drill motor function.

Farquhar, A., R, Fikes, W. Pratt and J. Rice (1995), "Collaborative Ontology Construction for Information Integration," Stanford University Technical Report KSL-95-63.

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

Goel, A., A. Gomez, N. Grue, J. W. Murdock, M. Recker and T. Govindaraj (1996), "Explanatory Interface in Interactive Design Environments," *Artificial Intelligence in Design '96*, J. S. Gero (ed.), Kluwer Academic Publishers, Boston.

Hardwick, M. and D. Loffredo (1995), "Using EXPRESS to Implement Concurrent Engineering Databases," " *Proceedings of the 1995 ASME Computers in Engineering Conference and the Engineering Database Symposium*, Boston, MA, September.

Hodges, J. (1997), "Multiple Levels in Mechanical Device Representation," *Intelligent Systems for Engineering: A Knowledge-Based Approach*, R. D. Sriram (ed.), Springer-Verlag, in press.

Iwasaki, Y. and B. Chandrasekaran (1992), "Design Verification through Function and Behavior-Oriented Representations: Bridging the Gap between Function and Behavior," *Artificial Intelligence in Design '92*, J. S. Gero (ed.), Kluwer Academic Publishers, Boston.

Kim, G. J., S. R. Gorti, A. Gupta, A. Wong and R. D. Sriram (1996), "An Object-Oriented Representation for Product and Design Process", Submitted to *Research in Engineering Design*.

Kim, T. S., S.-H. Han and Y. J. Shin (1996), "Product Data Management Using AP203 of STEP Standard," *Proceedings of the 1996 ASME Design Engineering Technical Conferences and Computers in Engineering Conference*, Paper No. 96-DETC/DAC-1069, Irvine, CA, August.

Little, A., K. L. Wood and D. McAdams (1997), "Functional Analysis: A Fundamental Empirical Study for Reverse Engineering, Benchmarking and Redesign," Proceedings of the 9th ASME International Conference on Design Theory and Methodology, Paper No. DETC97/DTM-3879, Sacramento, CA, September.

Russel, S. and P. Norvig (1995), *Artificial Intelligence: A Modern Approach*, Prentice Hall.

Shah, J. J., D. K. Jeon, S. D. Urban, P. Bliznakov and M. Rogers (1996), "Database Infrastructure for Supporting Engineering Design Histories," *Computer-Aided Design*, **28**(5).

Wong, A. and D. Sriram (1993), "Shared Workspaces for Computer-Aided Collaborative Engineering," MIT Intelligent Engineering Systems Laboratory, Document 93-06.

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