Capturing Empirically Derived Design Knowledge for Creating Conceptual Design Configurations

Tolga Kurtoglu, Matthew I. Campbell, Joah Gonzalez
Automated Design Lab
Department of Mechanical Engineering
University of Texas at Austin
Austin, Texas 78712-0292

Cari R. Bryant, Robert B. Stone, Daniel A. McAdams
Design Engineering Laboratory
Department of Basic Engineering
University of Missouri- Rolla
Rolla, Missouri 65409

ABSTRACT
In an ideal design process, designers envision a configuration of components prior to determining dimensions or sizes of these components. Given the breadth of suppliers and production methods that exist today, most engineered artifacts are a mix of both custom-made parts and OEM (original equipment manufacturer) parts. The design of any future artifact must be carefully planned to take advantage of the diverse set of possibilities. We conjecture that computational design tools could be developed to help designers navigate the design space in creating configurations from detailed specifications of function. In this research, a methodology is developed that extracts design knowledge from an expanding online library of components in the form of grammar rules. From an initial implementation of forty-five rules compiled from 15 components extracted from three products, we demonstrate a computational process that builds a new design configuration by borrowing concepts from how common functions are solved in related designs.

1 INTRODUCTION
The goal of this research is to develop a methodology and resulting computational tools that work with a designer in suggesting particular components that can be used to solve common functions within future products. This paper shows the details from an example (shown in Figure 1) in which rules are extracted from an electric knife, an electric coffee grinder, and an electric can opener and are used to build a configuration for a deli slicer. Unlike other research that attempts to automatically synthesize a design configuration, this research leverages an expanding online library of components from which design rules are developed to capture the knowledge of the original designer’s intent. The resulting computational design tool will help designers navigate the design space by suggesting components that can be used to fulfill the various function specifications.

Figure 1: In this paper, the rules derived from an electric knife, a coffee grinder, and a can opener are used to derive the configuration for deli slicer.
In following a general design process such as that proposed in Pahl and Beitz [1], one can see that many early design stages including task planning and clarification, and conceptual design are very much human design activities that lack generic methods that are effective in all problem domains. The success of a design process and its resulting artifact are subject to the experience and talents of the engineers involved. In this research, we hope to provide a method that may eventually help designers in developing conceptual solutions by banking on the success of past designs through the use of an online repository of design and their components.

Given the overwhelming number of original equipment manufacturers, such a computational tool would both increase the efficiency of the design process and the rigor of creating new solutions. Furthermore, the rules created from the repository could be initiated in a computational search process to propose complete solutions to a functional specification. In our paper from last year’s DTM [2] (also in an expanded journal article [3]), we show how a set of 69 grammar rules are developed to guide the design process from initial functional goal to a detailed function structure. In this work, we start with a function structure and find solution principles for the various functions of the function structure. From an initial implementation of forty-five rules compiled from 15 components extracted from three products, we demonstrate a computational process that builds configurations by borrowing from how components have been utilized in other designs as is shown in the design of the deli slicer in Figure 1.

In the remainder of this paper, we will layout the related work that has lead to this research (Section 2 and 3), followed by a description of the methodology (Section 4) and the implementation (Section 5). This is followed by a detailed description of the electric knife example (Section 6), and a discussion of the implications of this example and our future research goals (Section 7).

2 RELATED WORK

In recent years, engineering researchers have discovered that shape grammars, which were originally used in architectural research [4], provide a flexible yet ideally structured approach to engineering design methods [5]. A shape grammar is a set of shape rules that apply in a step by step way to generate a set, or language, of designs. Grammar based design systems offer the option of exploring the design alternatives as well as automating the design generation process. The concept of a grammar is that an experienced designer can construct a set of rules to capture his/her knowledge about a certain type of artifact. Ideally, the grammar can be constructed such that any execution of the rules creates a feasible solution [6] or captures the style of a specific period [7] or a specific designer [8].

An important offshoot of the shape grammar research is graph grammar research. Similar to production systems in cognitive psychology [9], graph grammars are comprised of rules for transforming nodes and arcs within a graph. These techniques create a formal language for generating and updating complex designs from a simple initial specification, or seed. Graph grammars are an emerging concept in design synthesis [10, 11, 12, 13]. The development of these rules encapsulate a set a valid operations that can occur in the development of a design. Such representations can produce a wider variety of candidates since solutions need not have common characteristics, but merely a common starting point.

Our approach to creating such grammar rules is to observe the common uses of components found in the repository. This encapsulation of design knowledge is similar to a variety of artificial intelligence approaches to knowledge representation. The representation is crucial because it establishes the limits of the search space and the scope of design complexity. Conceptual design as accomplished by human designers operates on a level not limited by number of components, types of components, or types of component configurations. Various artificial intelligence approaches have been adapted to engineering to mimic the open-endedness of real design problems, most notably the Kuipers QSIM approaches [14] and the SketchIT design tool [15]. There are also a number of generative methods [16, 17, 18], linguistic approaches to representing design function [19], and descriptions of how components influence one another [20, 21, 22, 23]. Finally, previous work by the authors in the A-Design research [24] shows a qualitative understanding of components that allow configurations to be created in either series or parallel. In this manner, the representation remains unstructured so that complex and conceptual design problems can be accomplished.

3 DESIGN KNOWLEDGE REPOSITORY

Cataloging the variety of electromechanical components has recently become an intense research area motivated by the enabling expansiveness and self-organization of the internet. Most prominently, NIST [25, 26, 27] has developed a set of information models to be used for modeling product knowledge at varying levels of detail. Specifically, this work established several fundamental classes for components such as intended functions, transfer functions, flows, forms, materials, behaviors, references, and constraints.

Part of the difficulty in capturing such component information is the lack of accepted definitions and divisions for function or purpose for a component. Efforts at University of Texas and now at University of Missouri-Rolla have sought a standard vocabulary for describing the functionality systems in the electro-mechanical domain [28]. The definition of function refers to a “blackbox” transformation of input flows into output flows for example a motor “converts electrical energy to mechanical energy.” Through numerous product dissections the research team converged to a manageable set of function names (Table 1) and their respective flows (Table 2) that comprise a canonical functional vocabulary.

Recent progress has established the concept of an Enhanced Bill of Materials (EBOM) that handles entry, management and export of repository knowledge. Currently, the EBOM concept is implemented in a JSP (JavaServer Pages) database. The
XML from the database server can be viewed as HTML through a standard web browser [29, 30] as shown in Figure 2. In its current format, the web-based system (http://function.basiceng.umr.edu/repository/) supports the browsing, searching, and exporting of design information. Given this format, our research has leveraged the repository data to extract design grammar rules that indicate popular ways of fulfilling specific functions by specific solution principles or components. As has been similarly done to establish a basis for functional names and flows, we have also sought an accepted set of component names. Through a similar dissection of products, we have converged to a set of 92 component names. Table 3 shows a partial list of these names which are further described in [31].

### 4 CONSTRUCTING DESIGNS FROM FUNCTION STRUCTURES AND REPOSITORY

In this section we present our approach for developing graph grammar rules from function structures and a web-based artifact repository. Grammars provide an effective method to generate design configurations through the execution of rules

---

**Table 1. Function examples from the functional basis (Hirtz et al., 2002).**

<table>
<thead>
<tr>
<th>Class</th>
<th>Basic</th>
<th>Class</th>
<th>Basic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branch</td>
<td>Separate</td>
<td>Control</td>
<td>Actuate</td>
</tr>
<tr>
<td></td>
<td>Distribute</td>
<td>Magnitude</td>
<td>Regulate</td>
</tr>
<tr>
<td></td>
<td>Import</td>
<td></td>
<td>Change</td>
</tr>
<tr>
<td></td>
<td>Export</td>
<td></td>
<td>Stop</td>
</tr>
<tr>
<td></td>
<td>Transfer</td>
<td>Convert</td>
<td>Convert</td>
</tr>
<tr>
<td></td>
<td>Guide</td>
<td></td>
<td>Support</td>
</tr>
<tr>
<td>Connect</td>
<td>Couple</td>
<td>Provision</td>
<td>Supply</td>
</tr>
<tr>
<td></td>
<td>Mix</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table 2. Flow examples from the functional basis (Hirtz et al., 2002).**

<table>
<thead>
<tr>
<th>Class</th>
<th>Basic</th>
<th>Class</th>
<th>Basic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Material</td>
<td>Human</td>
<td>Energy</td>
<td>Mechanical</td>
</tr>
<tr>
<td></td>
<td>Gas</td>
<td>Acoustic</td>
<td>Pneumatic</td>
</tr>
<tr>
<td></td>
<td>Liquid</td>
<td>Biological</td>
<td>Radioactive</td>
</tr>
<tr>
<td></td>
<td>Solid</td>
<td>Chemical</td>
<td>Thermal</td>
</tr>
<tr>
<td></td>
<td>Plasma</td>
<td>Electrical</td>
<td>Electromagnetic</td>
</tr>
<tr>
<td></td>
<td>Mixture</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Signal</td>
<td>Status</td>
<td>Hydraulic</td>
<td>Magnetic</td>
</tr>
<tr>
<td></td>
<td>Control</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**Figure 2: The design repository web interface.**
that create feasible solutions to the design problem. In this research, feasible solutions are represented by a configuration flow graph or CFG. The CFG shows individual components of a design, how the components are connected and the flows between them as shown in Figure 3. As a whole, it captures the conceptual solution of a design problem.

Our methodology is based on constructing a component flow graph given the function structure for a conceptual design. In this regard, our approach is unique in that a new graph (CFG) is constructed based on a separate, initially completed graph (i.e. the function structure). To perform this graph transformation, our grammar rules are defined to add components to the CFG that maintain a valid connection of components as well as meet specific function requirements specified with the function structure. Each of the 45 rules developed thus far is comprised of a left hand side (LHS) and right hand side (RHS) as shown in the examples of Figure 4.

In detail, the transformation from the function structure to CFG is part of the recognize-choose-apply cycle shown in Figure 5. Given an initial function structure, S, we first recognize all possible rules that have their LHS as a subset of S. This recognition step is followed by a choosing of one of the valid options. Currently, the options are presented to the user who then selects one of the rules to apply. In application, the CFG is updated as per the instructions provided in the RHS. Note that the same right-hand side could have multiple left-hand side alternatives. These cases are represented as separate rules. This ensures enumeration of different component alternatives for a given sub-function or set of sub-functions.

Our approach to developing these rules is based on design knowledge extracted from the aforementioned design repository. Through systematic product teardowns and data gathering in the repository, we are able to capture an existing product’s CFG and its function structure. Table 4 shows some of the data provided from the repository. The first column lists components from our agreed upon list of component names shown in Table 3 while the second column lists a more specific name for each component extracted from a product. The third column is determined by researchers matching components to functions within their agreed upon function structure. By examining the data in this table we carefully construct rules that capture the ways in which components fulfill various functions. The open-endedness of the grammar formulation allows us to escape the tendency to assign single components to single functions. Rules may insert multiple components for a single function, or a single component for multiple functions, as is the case in function sharing. This construction of rules is difficult since sometimes a rule may render another rule obsolete or invalid. Furthermore, the rules are also constructed so that when all the functions are fulfilled no more rules are recognized, thus indicating that the design is complete. The complete list of 45 rules is shown in detail at http://www.me.utexas.edu/~adlab/cfg_grammar.htm.

Table 3. A partial list of components used in this research.

<table>
<thead>
<tr>
<th>Abrasive Fan</th>
<th>Circuit Board</th>
<th>Lever</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actuator</td>
<td>Tube</td>
<td>Link</td>
</tr>
<tr>
<td>Antenna</td>
<td>Gear</td>
<td>Power Cord</td>
</tr>
<tr>
<td>Battery</td>
<td>Coupling Fastener</td>
<td>Heat Exchanger</td>
</tr>
<tr>
<td>Bearing</td>
<td>Latch Release</td>
<td>Heating Element</td>
</tr>
<tr>
<td>Belt</td>
<td>Material Filter</td>
<td>Housing</td>
</tr>
<tr>
<td>Blade</td>
<td>Electric Conductor</td>
<td>Handle</td>
</tr>
<tr>
<td>Cam</td>
<td>Condenser</td>
<td>Guide</td>
</tr>
</tbody>
</table>

Figure 3: The configuration flow graph (CFG) of a simple electric knife. Nodes of the graph represent product components, whereas arcs represent flows.

Figure 4: Example grammar rules: (a) recognizes the function “convert EE to RME” in the function structure and inserts a “motor” into the CFG. (b) recognizes the functions “import human”, “activate EE” and “export human” in the function structure and inserts a switch into the CFG.

Figure 5: Building the configuration flow graph from a function structure using the rule set.
5 IMPLEMENTATION

An algorithm has been created and programmed in C++ using Metrowerks CodeWarrior™ IDE Version 5.1 that follows the flowchart shown in Figure 6. The program first reads the function structure from a text file and initializes the configuration flow graph. At this stage, the CFG only has the input and output flows from the function structure with no components. The program then executes a three-step loop to implement the rules and updates the CFG by adding new component(s) in each iteration. The user interacts with the program via a simple dialog window, through which user’s choices for the rule(s) to be applied can be entered.

The program manages the rules and their applications until no further rules can be applied, thus terminating after a complete design configuration has been built for the given functional description. At the end of the process, it writes the resulting CFG to an output file. Currently, the CFG is encoded as a text file. Our future work efforts include incorporating a graphical based user interface that would display the CFG as a graph.

5.1 Rule Processing

The three steps of the rule processing are explained in the following sections.

5.1.1 Recognizing the Rules

A major issue in the implementation of the rules is the recognition as to where and when they can be applied. Thus, the first step in rule processing is recognizing the rules. At any instant, there are a number of rules that can be applied and the recognition algorithm determines all of the possible rules as well as their locations with the configuration of CFG. This is accomplished by traversing the list of 45 rules and checking each for applicability.

Rule recognition is twofold. First, a sub-function or a set of sub-functions is recognized from the function structure for which a component solution has not been assigned. Then, the CFG is scanned to recognize the location, i.e. head of a flow,
of where the rule can be applied. In addition, certain directionality is also imposed as part of the rule recognition. According to this, designs are built in an orderly fashion from inputs towards the outputs of the CFG. Any rule that satisfies these three conditions is then listed as a “recognized” rule along with its corresponding location.

5.1.2 Choosing the Rule to be Applied
In the second step, recognition is followed by the selection of the rule to be applied. The rule that is actually applied can be a choice of either the user or an automated process. In our current implementation, the user sequentially makes a selection from a list generated during the recognition step. The dialog window used for rule selection is shown in Figure 7.

5.1.3 Applying a Rule
The third and last step of rule processing is applying a selected rule. After the user makes a selection from the recognized rules list, the program updates the design to a new configuration by adding a component to the CFG as described by the selected rule. It then labels the sub-functions of the function structure that were part of the applied rule. This ensures that a sub-function or a set of sub-functions is not recognized again, once they have been assigned a component solution.

6 RESULTS
In this section, we illustrate the implementation of our grammar for two test cases.

In the first test case, we run the program after rules have been defined using the first product: an electric knife. The objective is to verify that the original design configuration of the electric knife can be created using the grammar. The function structure that was used as input to the program is shown at the top of Figure 8 which presents snapshots of the CFG development. For simplicity, the CFG’s are superimposed on the function structure to illustrate the mapping from sub-functions to components. In between the snapshots, we list the grammar rules that are applied. As can be seen from Figure 8, the design configuration is updated with a new component after

Figure 7: Screenshot of user interface

Figure 8: A pictorial representation of building the configuration flow graph (CFG) from a function structure using the rule set. Original design configuration of the electric knife is created using the grammar.
application of each new rule until no further rules can be recognized. The result is the completed design configuration shown at the end of the figure. As expected, the implementation of grammar rules successfully creates the original design configuration of the electric knife.

Before running the second test case, we extend our rule base by defining rules from two additional products: an electric can opener, and an electric coffee grinder. We then run the program using the same function structure as in the first test case.

This time, the program creates multiple design configurations, including the original design configuration of the electric knife. Figure 9 presents snapshots of a second CFG along with the numbers of the rules applied. In creating this alternative design configuration, the program successfully integrates component concepts from all three products into one complete concept variant. For example, in the original electric knife design, the rotational mechanical energy output of the electric motor is transmitted to the blades through a worm gear pair and a blade fixture (modeled as a link) before it is used to cut the food. In the alternative design, the program suggests that a gear pair and an output shaft be coupled to the blade to fulfill the same set of sub-functions. In fact, this configuration is the same as the one used in the coffee grinder to grind coffee beans. Similarly, the alternative design suggests that a separate handle be used to interface with the hand of an user, as opposed to incorporating the interface into the main housing as is done in designing the electric knife. This concept is inherited from the can opener, where a separate handle is used as a lever to activate the electric switch. Figure 10 shows a picture of a potential product that could be designed by pursuing the alternative concept generated by the program.

Examples such as these are very promising, because they show how the grammar approach can be extended such that a variety of concepts can be developed from a functional description of a product by synthesizing component solutions together that have been successfully used in the design of past products.
large number of complete solutions is generated first and pruned later based on feasibility criteria. Secondly, potential solution bias is eliminated. Designers with varying levels of experience in different domains usually rely on solutions derived from domains where they feel the strongest. By exploring a vast space of past design solutions, our method can generate design configurations by combining solution principles from different domains. Thirdly, the likelihood of success in design is increased by use of our method, simply because the methodology itself is based on leveraging the successes of the past.

Future work is aimed at expanding our rule set. Currently we are studying a dozen other products to increase the number of components and solution principles in our knowledge base. Also, we are exploring ways to develop an evaluation method, which will allow a designer to sort or rank generated design concepts. This could be done by introducing various evaluation criteria such as number of parts, estimated cost of design, ease of assembly, etc. Moreover, our research is exploring ways to evaluate the behavior of the generated design configurations by means of dynamic simulation.

Another future goal is to accommodate structural design specifications in addition to functional specifications. Function structures are widely accepted as a functional representation; however, they are limited in their ability to represent structural aspects of design. Towards that goal, we are developing a graphical representation called the design assembly model (DAM) that would complement configuration flow graphs (CFG’s) and facilitate the addition of structural components and interfaces into the conceptual design configurations. Finally, since the library of components would ideally be based on a dynamic online repository, the creation of rules would also ideally be created dynamically in real time. The current set of 45 rules has been created through the careful deliberation that is typical of creating any grammar rule set. So, an interesting extension to the work presented here would be a method to develop these rules automatically.

REFERENCES


Fenves, S., 2001, A core product model for representing design information, NISTIR 6736.


