A computational theory for the generation of solutions during early conceptual design

Cari R. Bryant

A COMPUTATIONAL THEORY FOR THE GENERATION OF SOLUTIONS DURING EARLY CONCEPTUAL DESIGN

by

CARI RIHAN BRYANT

A DISSERTATION

Presented to the Faculty of the Graduate School of the

UNIVERSITY OF MISSOURI–ROLLA

In Partial Fulfillment of the Requirements for the Degree

DOCTOR OF PHILOSOPHY

in

MECHANICAL ENGINEERING

2007

Robert B. Stone, Co-Advisor	Daniel A. McAdams, Co-Advisor
Matthew I. Campbell	Ashok Midha
David C. Drain	Xiaoping Du

DEDICATION

I dedicate this to my husband Scott and my parents Allen and Deon.

ABSTRACT

Advancement in technology is usually made by building on previous experiences and learning from past successes and failures. However, knowledge transfer in the broad field of product design is often difficult to accomplish. Research has shown that successful component configurations, observed from existing products, can be dissected and stored for reuse; but few computational tools exist to assist designers during the conceptual phase of design. Many well-known manual methods (e.g. brainstorming, intrinsic and extrinsic searches, and morphological analysis) rely heavily on individual bias and experience and are often time intensive, laborious tasks that may not catch solutions that are functionally analogous, but seemingly unrelated.

This research presents an automated concept generation tool that augments traditional activities during the conceptual phase of design. The automated concept generator draws on the existing knowledge contained within a repository of existing design solutions to quickly produce numerous feasible concepts early in the design process that each satisfy the functional requirements for a design problem. The computational algorithm enables the development of a computerized design tool that complements other concept generation activities, such as brainstorming and morphological analyses. By quickly presenting numerous concepts from products that have already been developed, this design tool provides a broader set of initial concepts for evaluation than a designer may generate alone when limited by his/her personal experiences.

ACKNOWLEDGMENTS

I would like to thank my advisors, Dan McAdams and Rob Stone, for their tireless support and dedication to mentoring me. I would also like to thank Matt Campbell for his willingness to travel to the far reaches of Missouri for my exams. I appreciate the time dedication and insightful questions and comments from all of my committee members. I also thank Ron Fannin for graciously allowing me to access his wisdom throughout the past years. Finally, I would like to thank my family. Without their constant support and encouragement, my achievements would be greatly diminished.

While working toward my Ph.D., I was supported by the National Science Foundation under grants IIS-0307419 and IIS-0307665. I am also grateful for the support I received from the University of Missouri–Rolla as a Curator’s Fellow.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

1. INTRODUCTION

Product design involves the transformation of a set of established requirements into a physical device capable of satisfying those requirements. The early stages of design, especially the stages involving concept generation and evaluation, are notoriously difficult to study because much of the processes occur subtly within a designer’s head. Because of the challenges associated with automating such an intrinsic and nuanced task, designers are faced with complex decisions to make during these early stages of design with few tools available to help manage the process. Designers can often feel overwhelmed by the idea of generating not just one but a broad array of solutions to a given design problem, especially if external pressure is being applied to develop an appropriate solution quickly. This combination of discomfort and pressure usually leads a designer to hastily rush through this critical phase of the design process. Consequently, designers, especially those with less experience, often fail to pursue and adequately evaluate an appropriate number of alternatives before selecting a design to embody. The research presented here seeks to support a designer during the conceptual phase of the design process with an automated tool capable of quickly searching a large database of design knowledge and delivering multiple relevant and easily identifiable conceptual solutions for a designer to pursue.

1.1. PRODUCT DESIGN OVERVIEW

Figure 1.1 diagrams common activities a designer must go through during the development of a design—from clarifying the needs the device must fulfill through generating the detailed embodiment of its structure. Many structured methods have emerged to help guide designers during the various stages of the design process (Pahl and Beitz, 1996; Otto and Wood, 2001; Ulrich and Eppinger, 1995). In particular, the systematic approach of Pahl and Beitz (1996) and Hubka and Eder (1984), representing European schools of design, has spawned many variant methodologies in American design literature (Ulrich and Eppinger, 1995; Ullman, 1997; Schmidt and Cagan, 1995; Pimmler and Eppinger, 1994; Shimomura, *et al.*, 1996; Cutherell, 1996; Otto and Wood, 1996, 1997). These methodologies (e.g. Pahl and Beitz, 1996 and Otto and Wood, 2001) take a designer through a specific set of steps devised to help dissect a design problem and build conceptual solutions based on the functionality that a product needs to exhibit. Functional modeling methods directly extract the functionality a solution must fulfill from the established customer needs, ideally removing designer biases that may be introduced by focusing on specific solutions too early in the design process. This act of abstraction helps a designer generate more creative and complete conceptual solutions and balance design choices between different components with the same functionality (Pahl and Beitz, 1996).

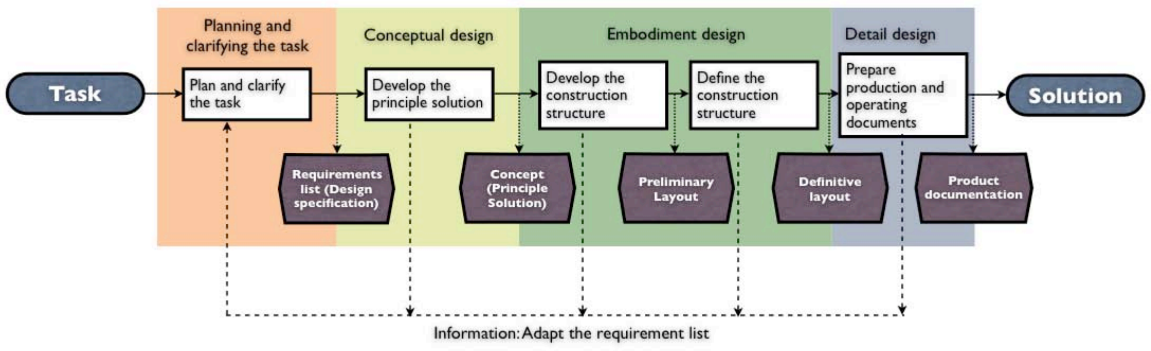

Planning and clarifying the task
Conceptual design
Embodiment design
Detail design
Task
Plan and clarify the task
Develop the principle solution
Develop the construction structure
Define the construction structure
Prepare production and operating documents
Solution
Requirements list (Design specification)
Concept (Principle Solution)
Preliminary Layout
Definitive layout
Product documentation
Information: Adapt the requirement list

Figure 1.1. Steps of the planning and design process (adapted from Pahl and Beitz, 1996.)

1.2. PROBLEM DESCRIPTION

Although systematic methodologies exist, Ivashok (2004) discusses the resistance designers seem to have toward applying them to generate initial design solutions and states that designers tend to quickly descend on potential solutions as a means to further define and understand a design problem. However, despite the tendency of designers to be resistant in employing rigid methods early in the design process (Cross, 1994), evidence also supports the idea that structured approaches can be helpful to students and demonstrates a positive correlation between a structured approach and both the quantity and quality of student designs (Radcliffe and Lee, 1989). Additionally, during the early stages of the design process, designers tend to focus more on loose representations of conceptual ideas, such as sketches and short descriptions, in order to begin to define a design solution. Yang (2003) concludes that, in the context of student design teams, it is important to generate and solidify a large number of ideas as well as begin prototyping a design early in the design process. These ideas seem to point toward the need for the seemingly tedious stages of systematic design to employ some level of automation to help integrate the benefits of a structured method with the more natural activities of a designer—a need that is most evident during the early phases of conceptual development.

The fuzzy front end of the conceptual design process has seen few attempts at automation, perhaps due in part to the evolving strategies and methodologies that exist for this phase of design. Many non-computational methods exist (e.g. techniques designed to stimulate creative solutions (Glover, *et al.*, 1989; de Bono, 1970) or use design rationale from successful designs (Navinchandra, *et al.*, 1991; Altschuller, 1984; Suh, 1995)) but do not employ any automated tools to help guide a designer. Furthermore, redesign tools (e.g. Quality Function Deployment (Prasad, 1998) and Life Cycle Analysis (van den Berg, *et al.*, 1995)) may prove initially confusing to an inexperienced designer. Computational tools that support the conceptual stage of design do exist, but often these tools address areas that support other aspects of conceptual design such as initial requirements gathering (e.g. organizational tools such as the TikiWiki project (Wodehouse, *et al.*, 2004)), the creation of function structures (e.g. the function grammar tool developed by Sridharan and Campbell (2005)), or optimization of well-established concepts (e.g. (Du and Chen, 2004)) rather than the generation of design solutions from existing design knowledge. Computational tools have been developed to assist a designer during the transition between defining a design's function and establishing its form, but often these approaches either limit the scope of design problems they are applicable to (e.g. Yates and Beaman, 1995; Hayes, 1995; Finkelstein, 1998), restrict concept generation to dynamic systems where a bond graph can be readily utilized (e.g. Welch and Dixon, 1991; Gui and Mäntylä, 1994; Bradley, *et al.*, 1993; Oh, *et al.*, 1996; Bracewell and Sharpe, 1996; Sieger and Salmi, 1997; Campbell, *et al.*, 1999, 2000, 2003), or utilize user interfaces that a novice designer may find difficult to interpret (e.g. Lu and Russomanno, 1999; Deng, 2002).

Conventional CAD programs are not designed to foster interactivity and creativity during the early stages of design (Akman, *et al.*, 1990), and suitable computational tools that support the fuzzy leading edge of the conceptual phase are still relatively young and underdeveloped. One area of research explores the development of computer tools that enable 2D designer sketches to be quickly transformed into 3D parameterized models, which can then be evaluated for the given design problem. Hearst, *et al.* (1998) state that computerized sketching research seeks to create an environment that encourages collaboration and modification in contrast to current computer interfaces that feel too formal and precise to stimulate creativity. However, computerized sketching tools (e.g. Lipson and Shpitalni, 2000; Qin, *et al.*, 2000; Eggli, *et al.*, 1997; Sturgill, *et al.*, 1995; Hwang and Ullman, 1990), although potentially useful, seem geared more toward capturing a designer’s ideas for further development early in the design process and do not seem to address the origination of the ideas to sketch. Other computer-aided conceptual design tools apply function-based associations to graphically describe the elements of a mechanical assembly (Serran and Gossard, 1992; Al-Hakim, *et al.*, 2000; Moore, *et al.*, 1997). Often, though, function and flow semantics are only assigned to a conceptual design after the structure has been chosen for manipulation by the software (e.g. the Multi User Groups research platform (Cera, *et al.*, 2002)), thus diluting any benefits that may be gained by first abstracting a problem.

1.3. MOTIVATION

An increasing emphasis is placed on generating the best design early in the design process as companies strive to reduce costs and develop more reliable designs with minimal environmental impact at all stages of a product's life cycle. This drive toward perfecting a design (both from a marketing and manufacturing standpoint) in its infancy requires that experience from completed designs be retained and reused so that accurate decisions are made as early as possible, when design changes cost the company much less, as shown in Figure 1.2. Often, this experience is retained in the mind of a tenured employee and is given as sagely advice to a fellow designer seeking counsel about a design decision. All too often, the tenured employee reaches retirement and the company is faced with either losing decades of valuable experience or spending significant amounts of money seeking consultation from the retired employee.

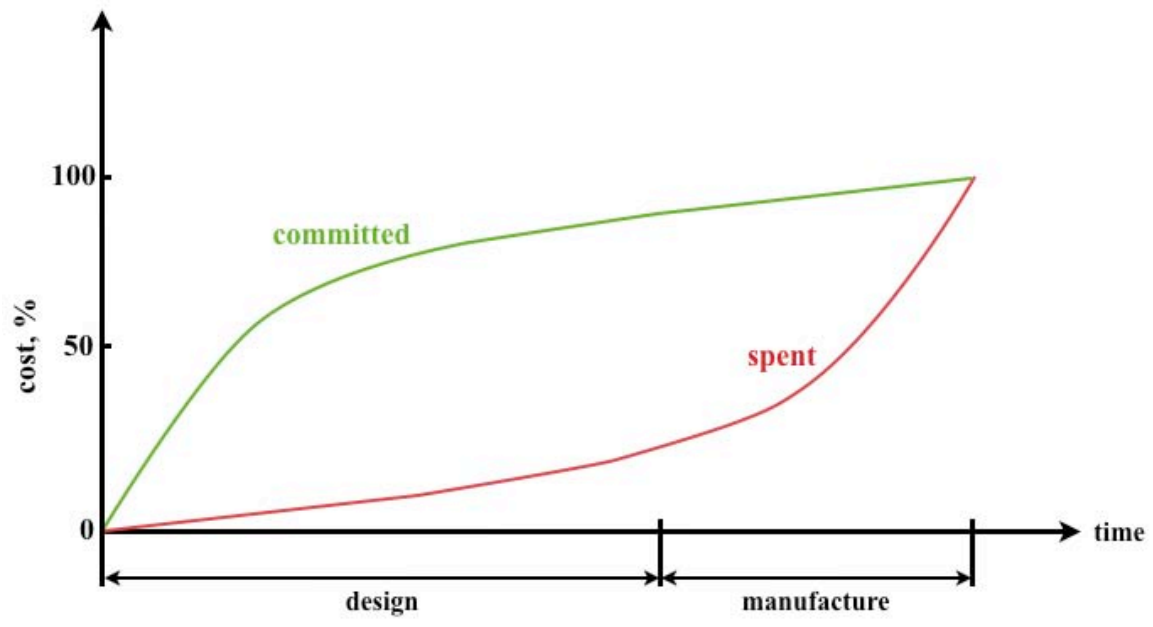


Figure 1.2. Allocated and expended costs during design development and manufacture. (adapted from Black, 1990).

Research has shown that successful component configurations, observed from existing products, can readily be dissected and stored for reuse (Bohm, *et al.*, 2005; Bohm and Stone, 2004). But, even if experience in the form of design knowledge is accessible, both the experienced and inexperienced designer may feel compelled to become fixated on a particular solution or domain restricted set of solutions based on instinct or, perhaps, a subconscious desire to pursue an initial ‘gut feeling.’ Designers traditionally have a limited number of options available to them to help generate multiple feasible design solutions to evaluate.

Many well-known manual methods for generating multiple ideas (e.g. brainstorming, intrinsic and extrinsic searches, and morphological analysis) are designed to stimulate a designer’s creativity but ultimately still rely heavily on individual bias and experience. Ideation is typically limited by experience, and experienced designers tend to pursue a larger array of solutions early in the design process. Structured design methodologies seek to encourage the generation and evaluation of a broad array of conceptual designs by leading a designer through a series of guided stages. When directed to generate multiple solutions for a given problem within a structured design process, designers with limited experience are often able to produce a few feasible concepts, but many of the ideas they produce are technically or realistically infeasible or merely minor variations of similarly themed solutions. Researching alternative solutions could potentially yield new concepts, but inexperienced designers are often still limited to searching for preexisting solutions to the same design problem. This raises the question of how one searches for something when (s)he does not know it exists?

Traditional methods for researching alternative ideas include interviewing more knowledgeable people, searching for relevant patents, performing an Internet (web) search, browsing through catalogs, or reverse engineering existing designs. Interviews may still be limited by the experiences of the person being interviewed, and the interviewee's biases may inhibit an inexperienced designer from pursuing a non-traditional solution. Patent databases, while vast, are not searchable in a manner that readily fosters innovation and often are only useful for focused searches into specific technologies. Web searches and catalogs are also vast sources of knowledge, but personal experience can severely narrow a designer's search, and the amount of available information may prove too daunting to effectively parse through. Reverse engineering existing products is potentially costly and time consuming and little information may be gained by dissecting only one or two products.

1.4. HYPOTHESIS AND OBJECTIVES

The challenge in creating useful conceptual design tools lies in finding innovative ways to help guide a designer toward the best solution(s) by building on existing design experience while simultaneously discouraging tendencies to make choices or evaluations based on hunches or biased methods. The following subsections elaborate on the research hypothesis driving this the research and the specific objectives accomplished by addressing this challenge.

1.4.1. Hypothesis. Using a database of stored design knowledge as a library, a computational design tool can be created that is capable of quickly generating multiple feasible solutions for a generic design problem. This study focuses on creating a design tool that integrates into a structured design methodology, transforms user-defined design specifications into a broad array of conceptual designs, and helps a designer evaluate the solutions returned. By quickly presenting numerous concepts from products that have already been developed, this design tool can provide a broader set of concepts to evaluate than a designer may generate alone when limited by his/her personal experiences.

1.4.2. Objectives. The objectives of this dissertation are to:

I. Create a computational theory for generating and filtering conceptual solutions for a design problem using designer-defined functional requirements and existing design knowledge mined from a repository of design knowledge.

II. Define a structured methodology for classifying the component terms used to communicate the generated conceptual solutions to the designer.

III. Implement the concept generator theory and algorithms as a software tool (hereafter referred to as the concept generator tool) to present feasible and relevant design alternatives to a designer. The software will be validated by:

   A. Showing the capability of the concept generator tool to produce design solutions comparable to those produced by upper level engineering students.

   B. Showing the capability of the concept generator to reproduce design solutions for existing products that do not directly comprise any of the information stored in the repository of design knowledge.

   C. Showing the capability of the concept generator to produce conceptual ideas for a new product design.

1.5. ORGANIZATION

The layout of this document is arranged to coordinate with the main objectives. First the reader is given a detailed look at the state of the art in design and especially conceptual design, as well as a detailed introduction to the specific existing design technologies that are used to support the automated design tool presented in this dissertation. The three chapters (Sections 3–5) following the background (Section 2) present the main contributions that this research delivers, namely a comprehensive generalized algorithm for the automatic generation of conceptual solutions and the technologies created to support its applied use. Later chapters present experiments and case studies performed to validate the research presented in Sections 3–5 and test the current limitations of the proposed technology.

Synopses of the included subject matter by chapter:

Section 2 gives the reader background information pertaining to all of the remaining chapters. This chapter covers an overview of the existing methodological framework and computational tools that form the base for the work presented in Sections 3–6. This chapter also covers the state of the art in computational design tools that support various stages of the design process.

Section 3 presents the algorithm that the concept generator uses to transform the user-defined input into conceptual ideas. A simple example using a finite repository of parts from a popular children's toy is included to help demonstrate how new conceptual solutions are created using the described algorithm,

Section 4 presents the software created from the algorithm described in Section 3. Two versions of the Java-based program are described; one with limited functionality that was initially used to test the efficacy of the algorithm in a real design scenario (described in Section 6), and one with expanded functionality capable of supporting a full graphical model of the requirements generated during a structured design process.

Section 5 presents the structured methodology created for classifying components under a proposed hierarchical structure, similar to ones used for the classification of living organisms. Additionally, function and port templates are proposed for each term to help establish a more rigorous structure for the inclusion of design data into the repository used to generate new conceptual solutions.

Section 6 presents a series of experiments and case studies performed to test the viability and usability of the proposed concept generator algorithm and implementation. Two studies involving independent student evaluations of an implementation of the computational tool are first presented. Finally, a case study investigating the effectiveness of the proposed computational tool is included.

The final section, Section 7, concludes the work presented, outlines the contributions made, and establishes future avenues of investigation that build on the research presented. Appendices and a list of References can be found immediately following Section 7.

2. BACKGROUND

2.1. INTRODUCTION

The following section begins with a review of the state of the art in conceptual design research and areas that support automated concept generation. In particular, we first review systematic approaches to conceptual design and then focus on specific product function and component representations and design knowledge collection techniques on which the automated concept generation theory is built. Finally, a review of existing computational tools that support designers throughout various stages of the design process is presented.

2.2. STRUCTURED DESIGN METHODS

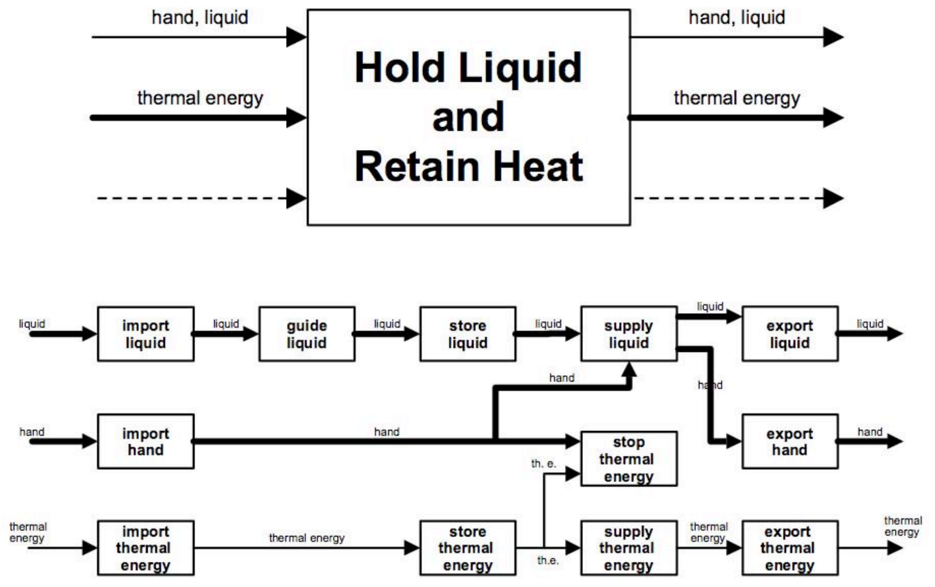
The fuzzy front end of the conceptual design process has seen few attempts at automation, perhaps due in part to the evolving strategies and methodologies that exist for this phase of design. However, over the past decade several methodologies have coalesced around the functional decomposition and partial solution manipulation techniques of Pahl and Beitz (Pahl and Beitz, 1988; Ulrich and Eppinger, 1995; Otto and Wood, 1996, 1997, 2001; Hubka and Ernst Eder, 1984; Ullman, 1997; Schmidt and Cagan, 1995; Pimmler and Eppinger, 1994; Shimomura, *et al.*, 1996; Cuthrell, 1996). Subfunction descriptions are necessary elements of a formal approach to identify or derive a functional model for a product (originally called a function structure by Pahl and Beitz (1988)) to initiate the concept generation phase of design. For this research, the functional model derivation method as prescribed in Stone and Wood (2000) and

Kurfman, *et al.* (2003) is followed. The procedure is comprised of a five-step method summarized as:

1. Identify input and output flows that address customer needs (or other high level requirements).
2. Generate a black box model (a model of the overall function and input/output flows) of the system that the performance model describes.
3. Create function chains for each input flow–i.e. “Be the flow” and imagine traversing through the system noting each operation (e.g. function) that is performed on you prior to leaving the system.
4. Aggregate function chains into a functional model.
5. Check that each customer need is addressed by at least one subfunction. If not, then add or adapt functionality to meet remaining customer needs.

To briefly illustrate this technique, the functional model of an insulating cup is shown in Figure 2.1. The black box model is constructed based on the overall product function and includes the various energy, material, and signal flows involved in the global functioning of the product. The detailed functional model is then derived from subfunctions that operate on the flows listed in the black box model.

<u>Customer Needs</u>
- Does not burn hand
- Comfortable fit in hand
- Keeps drink hot for extended time
- Easy to drink
- Holds a standard Starbucks latte
- …



(a) (b)

Figure 2.1. For an insulating cup, (a) a snippet of customer needs leads to a (b) black box and functional model of a cup following the functional model derivation method.

Functional models for any product can be generated using this technique. Repeatability, ease in storing and sharing design information, increased scope in the search for solutions, and tracking of input and output flows are some of the advantages of functional models (Pahl and Beitz, 1988; Stone and Wood, 2000). When following the functional model derivation method outlined above, designers in an experimental group used 50% fewer terms to describe the functionality of the same product, and increased the clarity of design communication (Kurfman, *et al.*, 2003). On average, the experimental group found 82% of the important subfunctions of a very detailed "control" functional model after only one training session, indicating repeatability but not an exact science.

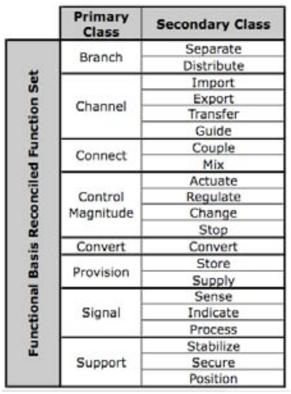
In many respects, any type of model creation to represent engineering systems is dependent on the skills and choices of the engineer. Different engineers are likely to report slightly differing results unless given extremely constrained scenarios. The same is true with functional modeling. In fact, this may be a strength of functional modeling as part of an original concept generation approach. The subtle differences between designers' models may promote the exploration of innovative alternatives. For the present purpose of conceptual design, designers who follow the functional model derivation method outlined above can generate functional models that are 'repeatable enough' to generate similar solutions with the concept generator algorithm presented in Section 3.

2.3. STANDARDIZED "LANGUAGES" IN DESIGN

Use of a standardized vocabulary is a beneficial tool for efficiently applying a computational method to a natural language process such as conceptual design. Two such existing vocabularies were utilized during the course of this research to facilitate the input of functional requirements and the output of conceptual solutions for a design. These two vocabularies, the Functional Basis of Design and the Component Basis, are described in the next two subsections.

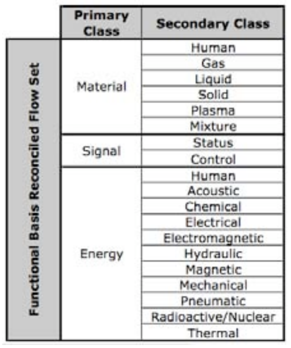
2.3.1. The Functional Basis of Design. The lack of a precise definition for small easily solved subfunctions has spurred research into the development of a high level design language (sometimes called a vocabulary or taxonomy) to describe product function and thus enable a systematic approach to functional modeling (Hundal, 1990; Koch, *et al.*, 1994; Malmqvist, *et al.*, 1996; Altshuller, 1984; Kirschman and Fadel, 1998; Kitamura and Mizoguchi, 1998, 1999; Umeda and Tomiyama, 1997; Sasajima, *et al.*, 1995). In order to make elements of the early design phases computable, researchers have continued to pursue a standard language that unifies the previous works (Otto and Wood, 1997; Little, *et al.*, 1997; Stone and Wood, 2000; Murdock, *et al.*, 1997; Szykman, *et al.*, 1999; Hirtz, *et al.*, 2002). The result of these recent efforts is a design language known as the Functional Basis. Shown in Table 2.1 and Table 2.2, the hierarchically arranged Basis terms are utilized during the generation of a black box model and functional model (discussed in Section 2.2) in order to encapsulate the actual or desired functionality of a product (Hirtz, *et al.*, 2002). In this approach, the designer follows a rigorous set of steps to define a new or redesigned product's functionality prior to exploring specific solutions for the design problem (Stone and Wood, 2000). The Functional Basis is intended to be broad enough to span the entire electro-mechanical design space without repetition and has been independently verified by other researchers such as Ahmed (Ahmed and Wallace, 2003; Ahmed, *et al.*, 2005) and Wood (Wood, *et al.*, 2005; Gietka, *et al.*, 2002). In Table 2.1, engineering functions are categorized as 8 primary classes that are further specified as secondary and tertiary (not shown) categories. In Table 2.2, engineering flows are categorized as three primary classes (material, signal and energy) and then further specified as secondary and tertiary (not shown) categories within each class.

Table 2.1. Function classes under the Functional Basis (for term definitions, see Hirtz, *et al.*, 2002).



	Primary Class	Secondary Class
Functional Basis Reconciled Function Set	Branch	Separate
		Distribute
	Channel	Import
		Export
		Transfer
		Guide
	Connect	Couple
		Mix
	Control Magnitude	Actuate
		Regulate
		Change
		Stop
	Convert	Convert
	Provision	Store
		Supply
	Signal	Sense
		Indicate
		Process
	Support	Stabilize
		Secure
		Position

Table 2.2. (Below) Flow classes under the Functional Basis (for term definitions, see Hirtz, *et al.*, 2002).



	Primary Class	Secondary Class
Functional Basis Reconciled Flow Set	Material	Human
		Gas
		Liquid
		Solid
		Plasma
		Mixture
	Signal	Status
		Control
	Energy	Human
		Acoustic
		Chemical
		Electrical
		Electromagnetic
		Hydraulic
		Magnetic
		Mechanical
		Pneumatic
		Radioactive/Nuclear
		Thermal

The black box model is constructed based on the overall product function and includes the various Functional Basis energy, material, and signal flows involved in the global functioning of the product. A detailed functional model is then derived using Functional Basis function terms that operate on the flows from the black box model. Repeatability, ease in storing and sharing design information, and increased scope in the search for solutions are some of the advantages functional models that incorporate the Functional Basis exhibit (Stone and Wood, 2000; Kurfman, *et al.*, 2001).

2.3.2. Component Basis. The component naming vocabulary employed throughout this research, termed the Component Basis, enhances the usefulness of the information contained within the design knowledge repository by grouping similar product artifacts into related classes (Kurtoglu, *et al.*, 2005, 2007). For example, specific instances of components in different products may be named "motor 1", "shaded pole induction motor", or "dc motor". Using the Component Basis, each of the of these components would be identified as similar and tagged as an "electric motor". Use of this vocabulary allows for groupings of similar components to be viewed as a single more abstract concept variant when returned as a result from a computational tool. Also, this clustering of like components helps eliminate redundancies that bog down computations. By eliminating these redundancies, a larger set of unique and more abstract concept variants can be quickly generated and evaluated using the proposed computational tool. After concept variants are selected using the generalized Component Basis names, individual artifacts classified under the chosen Component Basis names can then be more closely investigated to spur a more specific concept variant idea. For example, if a returned concept variant included an "electric motor", the design repository (described in detail in Section 2.4.) could be accessed to provide the designer with the specific examples "motor 1", "shaded pole induction motor", or "dc motor".

2.4. DESIGN KNOWLEDGE STORAGE AND RETRIEVAL

Functional models reveal functional and flow dependencies and are useful for capturing design knowledge from existing products. Over the course of several years, a web-based repository of design knowledge has been developed and refined at the

University of Missouri–Rolla and in collaboration with the University of Texas at Austin, Pennsylvania State University, Bucknell University, and Virginia Polytechnic Institute and State University (Bohm, *et al.*, 2004, 2005). This repository, which includes descriptive product information such as functionality, component physical parameters, manufacturing processes, failure modes, and component connectivity, now contains detailed design knowledge on over 100 consumer products and the components that comprise them (in total over 4500 design artifacts are currently included in the repository).

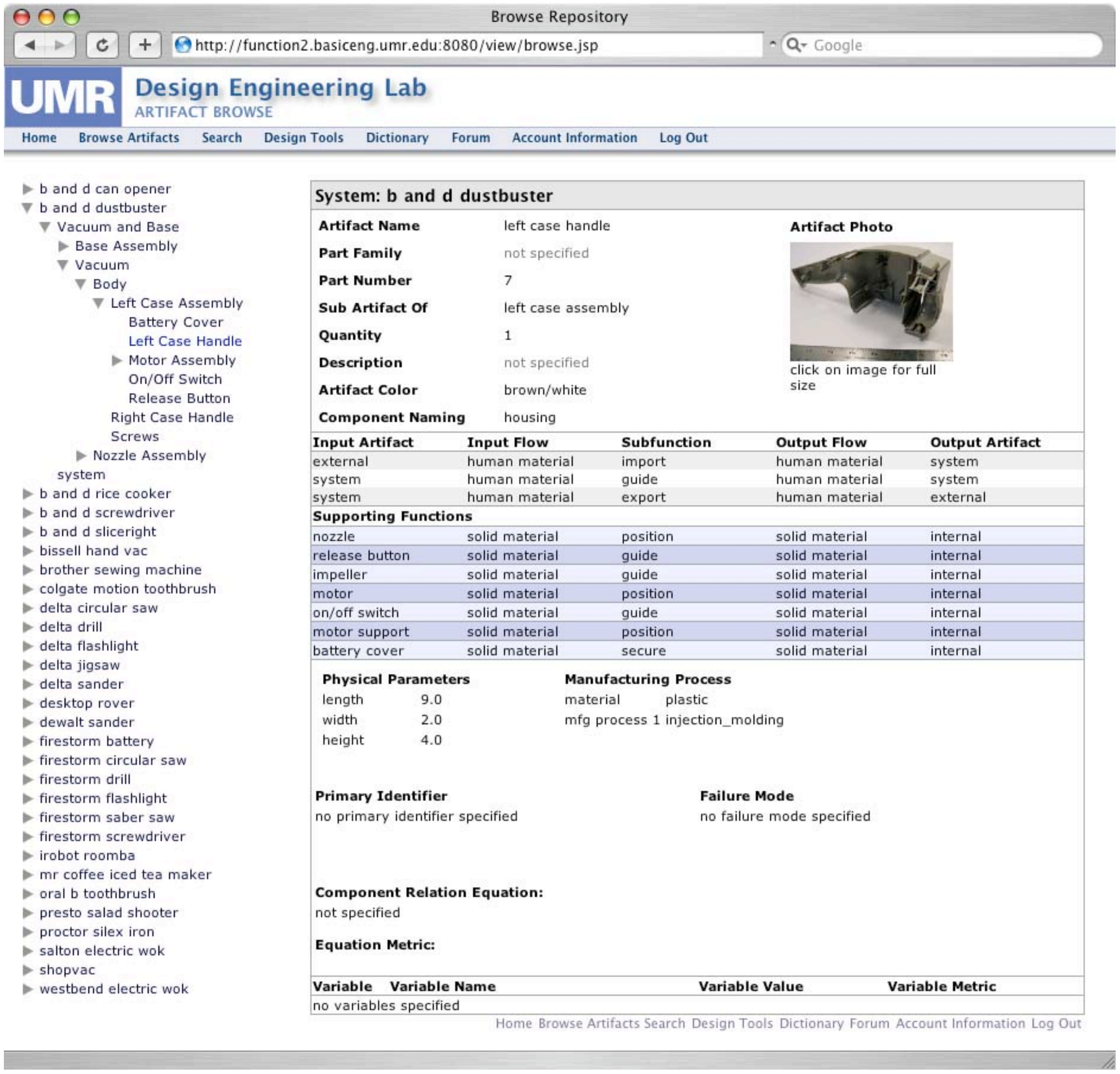
2.4.1. Information Captured by the Repository. Several design artifact attributes are captured when entering products into the UMR design repository database. These attributes are stored in a relational database where each record contains an *Artifact Name* as a free form text field where the user can define the name of an artifact, a *Part Family* as a free form text field that can be used to catalog similar artifacts as a type or family, and a *Part Number*, which is a sequential artifact number given when the artifact populates the database. Information about the actual function of an artifact is captured in the *Subfunction* field as a value from the Functional Basis described above.

Quantity, *Artifact Color*, *Manufacturing Process*, *Material*, and *Description* fields further describe the given aspects of the particular artifact along with fields to capture rough geometric dimensions. An *Assembly* field denotes whether or not the artifact is a composite assembly or atomic, and a *Supporting Function* field denotes whether or not the artifact is secondary to the product's operation. A *Component Nami*ng field references the list of standard component terms, described in Section 2.3.2., to abstractly identify the class of the artifact.

Artifact relationships are captured by the *Sub*-A*rtifact Of* field, which establishes a parent-child relationship; the *Input Artifact* and *Output Artifact* fields, which are used to trace flow from the current artifact to the corresponding input/output artifacts; and the *Input Flow* and *Output* Flow fields which similarly trace the input and output flows to other artifacts using values from the Functional Basis. With these fields, various relationships and connections can be drawn from the repository.

2.4.2. Using the Design Repository. The repository web interface, which offers guest and registered user access, is located at http://function.basiceng.umr.edu/repository/. The top-level options within the web repository are *Browse*, *Search*, *Design Tools*, *Design Methodology Dictionary* and *Account Information*. With the web-based repository, a user can browse and search artifacts, generate design tools, and view a dictionary of function and flow terms.

The *Browse* feature allows users to navigate through the repository. When *Browse* is initially selected, all of the high-level systems within the repository are shown at the left of the screen. The systems can be expanded such that artifacts within the system are exposed. A hierarchical menu system allows for systems to be expanded through subassemblies down to singular artifacts. The menu system draws information from the *Subartifact_Of* field of the database to establish artifact hierarchy. Finally, when an artifact or assembly is selected, a repository listing of the artifact is shown on the right portion of the screen. A screenshot of the *Browse* feature is shown in Figure 2.2.


Browse Repository
http://function2.basiceng.umr.edu:8080/view/browse.jsp
Google
UMR
Design Engineering Lab
ARTIFACT BROWSE
Home Browse Artifacts Search Design Tools Dictionary Forum Account Information Log Out

- ▶ b and d can opener
- ▼ b and d dustbuster
  - ▼ Vacuum and Base
    - ▶ Base Assembly
    - ▼ Vacuum
      - ▼ Body
        - ▼ Left Case Assembly
          - Battery Cover
          - Left Case Handle
          - ▶ Motor Assembly
          - On/Off Switch
          - Release Button
        - Right Case Handle
        - Screws
      - ▶ Nozzle Assembly
  - system
- ▶ b and d rice cooker
- ▶ b and d screwdriver
- ▶ b and d sliceright
- ▶ bissell hand vac
- ▶ brother sewing machine
- ▶ colgate motion toothbrush
- ▶ delta circular saw
- ▶ delta drill
- ▶ delta flashlight
- ▶ delta jigsaw
- ▶ delta sander
- ▶ desktop rover
- ▶ dewalt sander
- ▶ firestorm battery
- ▶ firestorm circular saw
- ▶ firestorm drill
- ▶ firestorm flashlight
- ▶ firestorm saber saw
- ▶ firestorm screwdriver
- ▶ irobot roomba
- ▶ mr coffee iced tea maker
- ▶ oral b toothbrush
- ▶ presto salad shooter
- ▶ proctor silex iron
- ▶ salton electric wok
- ▶ shopvac
- ▶ westbend electric wok

System: b and d dustbuster

Artifact Name	left case handle
Part Family	not specified
Part Number	7
Sub Artifact Of	left case assembly
Quantity	1
Description	not specified
Artifact Color	brown/white
Component Naming	housing

Artifact Photo

click on image for full size

Input Artifact	Input Flow	Subfunction	Output Flow	Output Artifact
external	human material	import	human material	system
system	human material	guide	human material	system
system	human material	export	human material	external
Supporting Functions				
nozzle	solid material	position	solid material	internal
release button	solid material	guide	solid material	internal
impeller	solid material	guide	solid material	internal
motor	solid material	position	solid material	internal
on/off switch	solid material	guide	solid material	internal
motor support	solid material	position	solid material	internal
battery cover	solid material	secure	solid material	internal

Physical Parameters

length	9.0
width	2.0
height	4.0

Manufacturing Process

material	plastic
mfg process 1	injection_molding

Primary Identifier
no primary identifier specified

Failure Mode
no failure mode specified

Component Relation Equation:
not specified

Equation Metric:

Variable	Variable Name	Variable Value	Variable Metric
no variables specified			

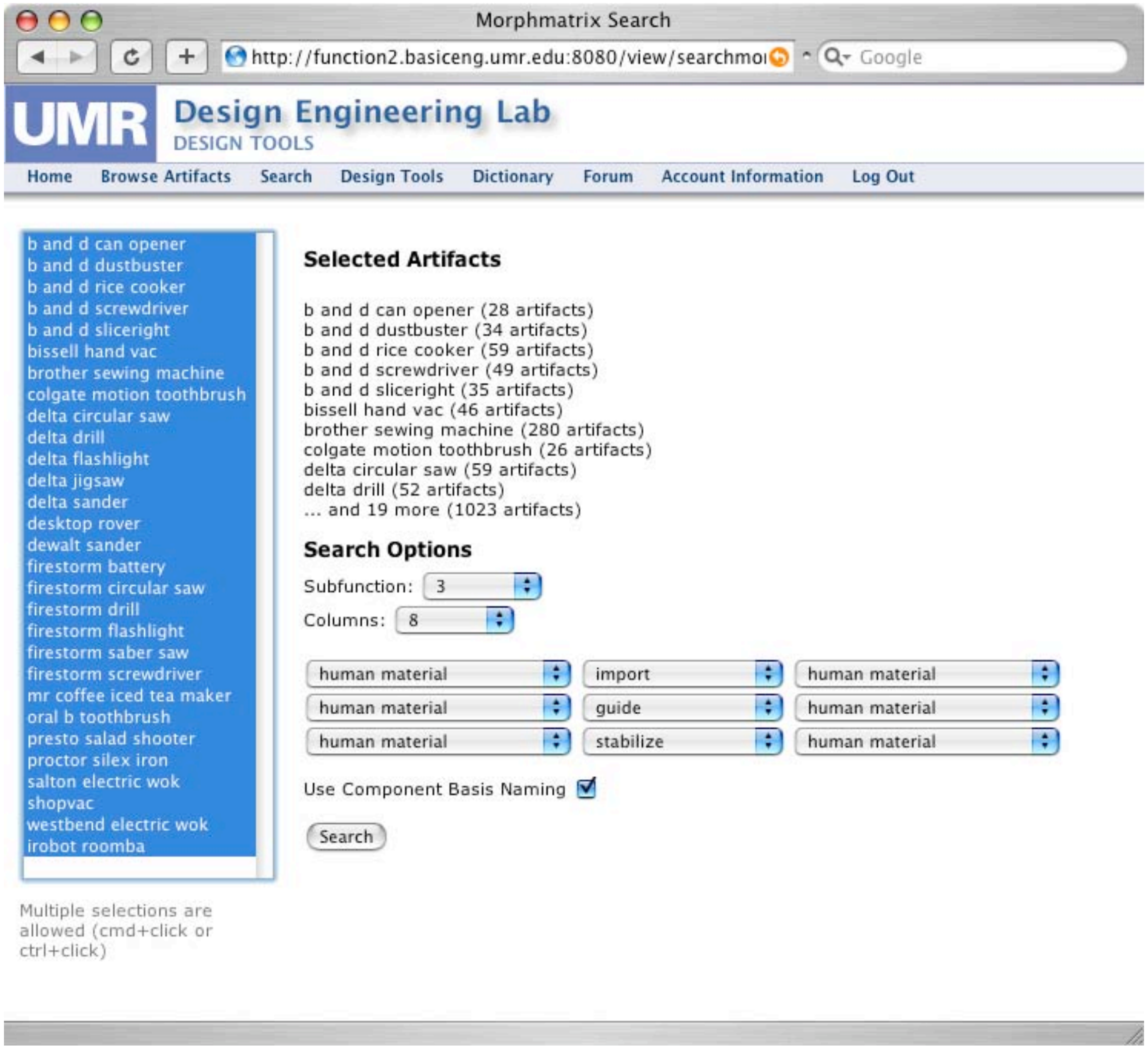
Home Browse Artifacts Search Design Tools Dictionary Forum Account Information Log Out

Figure 2.2. The UMR Design Repository web interface. Access to the repository may be requested at http://function.basiceng.umr.edu/repository.

When a repository user selects the design tool option, they are presented with a listing of the high-level systems contained in the repository and selection boxes to denote the type of desired design tool output. Once one or more systems are selected, a summary of the selected systems is presented, notifying the user of the number of artifacts within the systems and system descriptions. The repository can currently output function-component and design structure matrices as well as bills of materials. Because the design tools are not stored but rather created on demand directly from the repository database, the user will always be presented with the most up-to-date design tool.

2.4.3. The Web-Based Morphological Search. The morphological matrix introduced by Zwicky is a now a classic technique for use in conceptual design (Zwicky, 1969). This method provides the design engineer with a simple, albeit manual, means for bookkeeping potential physical solutions and their corresponding functionality. A morphological matrix is traditionally created by listing all of the subfunctions for a design and brainstorming solutions to each subfunction, listing the solutions as columns and the subfunctions as rows (Pahl and Beitz, 1996; Otto and Wood, 2001; Ulrich and Eppinger, 1995; Hubka and Eder, 1984; Ullman, 1997). In a manual engineering design context, the morphological matrix is limited to the concepts generated by the engineer, although the morphological matrix is one technique that can be used in conjunction with overall design processes such as 6-3-5 or the reverse engineering and redesign method of Otto and Wood (2001).

The web-based morphological search tool is an automated online tool that designers can use to filter and browse through the product knowledge contained within the web-based repository. Accessed through either a guest or personalized user account at http://function.basiceng.umr.edu/repository/, a designer may reach the design tool via a web-browser on any computer connected to the Internet. Upon logging into the design repository, the user is presented with an options menu. To perform a morphological search, the user navigates to the *Search* page and is presented with the option to perform either a “Standard Artifact Search” or a “Morphological Chart Search”. Once “Morphological Chart Search” is selected, the user is presented with the morphological search options shown in Figure 2.3.



b and d can opener
b and d dustbuster
b and d rice cooker
b and d screwdriver
b and d sliceright
bissell hand vac
brother sewing machine
colgate motion toothbrush
delta circular saw
delta drill
delta flashlight
delta jigsaw
delta sander
desktop rover
dewalt sander
firestorm battery
firestorm circular saw
firestorm drill
firestorm flashlight
firestorm saber saw
firestorm screwdriver
mr coffee iced tea maker
oral b toothbrush
presto salad shooter
proctor silex iron
salton electric wok
shopvac
westbend electric wok
irobot roomba

Multiple selections are allowed (cmd+click or ctrl+click)

Selected Artifacts

b and d can opener (28 artifacts)
b and d dustbuster (34 artifacts)
b and d rice cooker (59 artifacts)
b and d screwdriver (49 artifacts)
b and d sliceright (35 artifacts)
bissell hand vac (46 artifacts)
brother sewing machine (280 artifacts)
colgate motion toothbrush (26 artifacts)
delta circular saw (59 artifacts)
delta drill (52 artifacts)
... and 19 more (1023 artifacts)

Search Options

Subfunction: 3

Columns: 8

human material	import	human material
human material	guide	human material
human material	stabilize	human material

Use Component Basis Naming ☑

Figure 2.3. The morphological search input.

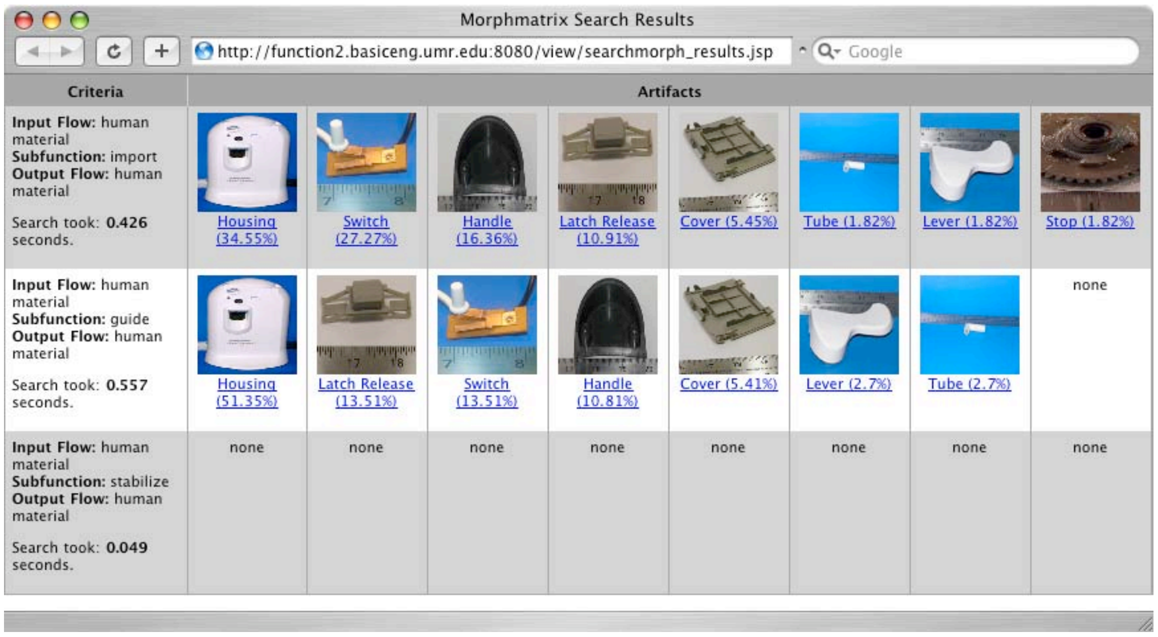
A list of available products is presented on the left hand side of the morphological search input. The user can select any combination of the products listed depending on their desired search domain. With the search base selected, the user then selects the number of subfunctions they wish to enter through the “Subfunction:” pull-down menu. At this time, a maximum of 10 subfunctions can be entered for a single search. If more than 10 subfunctions exist, the user must perform multiple searches. Once the number of subfunctions is selected, the user must specify the number of columns they wish to appear in the search return. A maximum of 20 columns can be displayed although 10 columns typically capture most, if not all, of the possible returns.

The user can now begin to specify the subfunctions they wish to search for by using the pull-down menus. Subfunctions are entered as a tuple representing the input flow, subfunction and output flow. The first subfunction entered in Figure 2.3 relates to “import human material” but is specified in the format (human material, import, human material). For most functions, the input and output flow are identical; however, the input and output flow for some functions (e.g. convert) are different.

With all of the desired subfunction tuples entered, the user can utilize the “Use Component Basis Naming” checkbox to choose how search results are returned. Checking the box categorizes returned artifacts under the Component Basis. Leaving the box unchecked will return results categorized by the name given to a specific artifact. For example, artifacts may be named “motor,” “electric motor” and “dc motor,” but they are all categorized by the Component Basis as “Electric Motor.” Choosing to categorize search results by the Component Basis will group all instances of an electric motor as

"Electric Motor." Without using the Component Basis categorization, the instances of "motor," "electric motor" and "dc motor" would be returned distinctly.

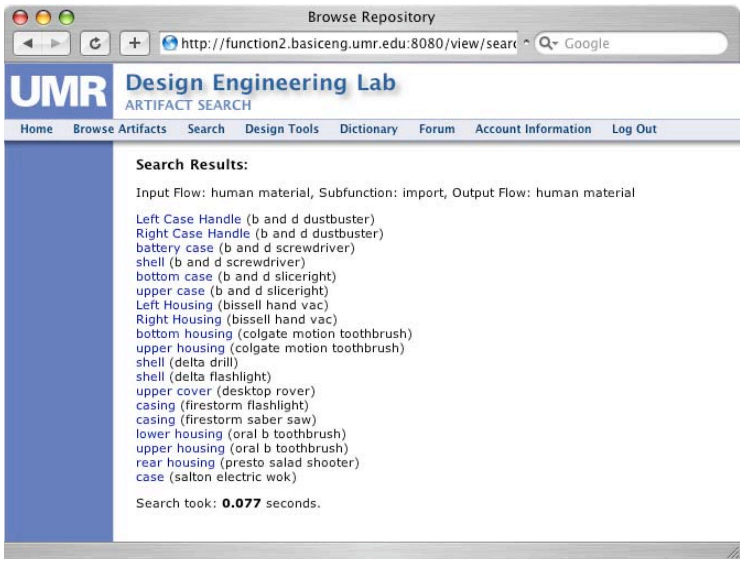
Upon submitting the search, a new browser window is opened containing the search results. These results for the three example subfunction tuples entered in Figure 2.3 are shown below in Figure 2.4. The left-most column of the results page displays the subfunction search criteria and subsequent columns (up to the amount specified) show the groupings of artifacts solving the given function. The results are sorted within each row by their rate of return. For example, a "Housing" of some sort is found to solve "Import Human Material" in 34.55% of the total number of solutions to "Import Human Material."


Morphmatrix Search Results
http://function2.basiceng.umr.edu:8080/view/searchmorph_results.jsp
Google

Criteria	Artifacts							
Input Flow: human material **Subfunction:** import **Output Flow:** human material Search took: **0.426** seconds.	Housing (34.55%)	Switch (27.27%)	Handle (16.36%)	Latch Release (10.91%)	Cover (5.45%)	Tube (1.82%)	Lever (1.82%)	Stop (1.82%)
Input Flow: human material **Subfunction:** guide **Output Flow:** human material Search took: **0.557** seconds.	Housing (51.35%)	Latch Release (13.51%)	Switch (13.51%)	Handle (10.81%)	Cover (5.41%)	Lever (2.7%)	Tube (2.7%)	none
Input Flow: human material **Subfunction:** stabilize **Output Flow:** human material Search took: **0.049** seconds.	none	none	none	none	none	none	none	none

Figure 2.4. The morphological search results.

For this particular search, results were returned for “import human material” and “guide human material” while no artifacts were found for the “stabilize human material” criteria. To view specific instances of a returned component grouping, the user can click on the link below the component image. Figure 2.5 shows all of the 19 artifacts classified as a “Housing” for the “import human material” search criteria. Listed along side each artifact is the artifact’s parent product. For example, the “Left Case Handle” artifact originated from the Black and Decker Dustbuster. If the user wishes to view more information about a specific artifact, they can do so by clicking the artifact name. The screen-shot seen previously in Section 2.4.2. as Figure 2.2 shows the Browse page that appears when “Left Case Handle” is selected. The Browse page shows additional information such as the additional subfunctions associated with the artifact, artifact color, material, manufacturing process, and physical parameters.



Search Results:

Input Flow: human material, Subfunction: import, Output Flow: human material

Left Case Handle (b and d dustbuster)
Right Case Handle (b and d dustbuster)
battery case (b and d screwdriver)
shell (b and d screwdriver)
bottom case (b and d sliceright)
upper case (b and d sliceright)
Left Housing (bissell hand vac)
Right Housing (bissell hand vac)
bottom housing (colgate motion toothbrush)
upper housing (colgate motion toothbrush)
shell (delta drill)
shell (delta flashlight)
upper cover (desktop rover)
casing (firestorm flashlight)
casing (firestorm saber saw)
lower housing (oral b toothbrush)
upper housing (oral b toothbrush)
rear housing (presto salad shooter)
case (salton electric wok)

Search took: **0.077** seconds.

Figure 2.5. Detailed component list for “housing”.

2.4.4. Downloadable Design Tools. The knowledge contained in the repository is steadily expanding and benefits from a broad base of consumer products. As indicated in Figure 2.6, design generation tools like the function-component matrices (FCMs) and design structure matrices (DSMs) can be readily created from single or multiple products using the web-browser interface. The downloadable matrices can be used in a variety of ways to enhance the design process (Bohm, *et al.*, 2004, 2005). FCMs contain information about the functionality of the components comprising the subset of products chosen for analysis.

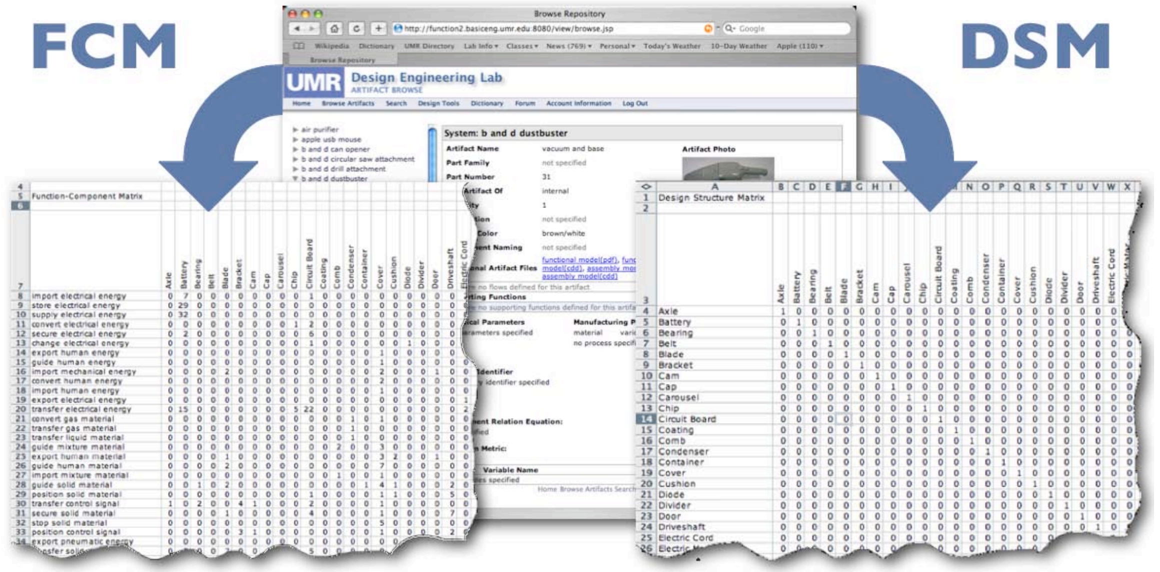

FCM
DSM
UMR Design Engineering Lab
ARTIFACT BROWSE
Function-Component Matrix
Design Structure Matrix
System: b and d dustbuster

Figure 2.6. From the web-based repository (center), a designer may extract information about component functionality in the form of a function-component matrix (FCM, left) and component compatibility in the form of a design structure matrix (DSM, right).

Each nonzero cell entry, x_{ij} in the FCM matches a component classification term j with a subfunction i that it had solved in a product that had previously been dissected and stored in the repository, where x_{ij} is the number of instances of the j^{th} component exhibiting the i^{th} functionality. For example, in the column labeled 'gear', common functionality includes the rows 'change mechanical energy' and 'transfer mechanical energy'. Similarly, each DSM generated from the repository contains component compatibility information for the components comprising the subset of products selected. In a DSM, positive compatibility between component j and component i is indicated when a 1 occupies the cell at d_{ij} (e.g. an electric motor and a gear). DSM cell entries set at 0 indicate that the corresponding row and column components were not directly connected in any of the products selected to generate the DSM. Each of these matrices is a simple but potentially powerful representation of the design knowledge from existing designs.

2.5. TOOLS TO SUPPORT THE DESIGN PROCESS

Many tools are available, both computational and manual, to assist a designer during various stages of the design process outlined in Figure 1.1. This section will provide an overview of design tool research that supports various stages of the design process and illustrates where the research presented in this dissertation fits into the process.

2.5.1. Idea Generation Techniques. Concept generation research has traditionally focused on developing methods that improve the quality and variety of concepts generated. These methods are often kept simple and efficient such that designers are not burdened by the details or limitations of the method. The most common concept generation method is known as brainstorming (Osborn, 1957). The term brainstorming is frequently applied to any idea generation technique. Brainstorming as a specific method requires a group of individuals to follow the basic rules of 1) avoiding criticism, 2) welcoming “wild ideas”, 3) building on one another’s ideas, and 4) preferring more ideas than dwelling on specific ones. A more structured concept generation method can be found in the techniques known as C-Sketch (Shah, 1998) and 6-3-5 (Rohrbach, 1969). The latter of these sketch-based methods requires six participants to independently create three ideas at a time in a series of five rounds. The added constraints of the method ensure that individuals participate equally, which may be more difficult to enforce in traditional brainstorming.

In addition to these team-based methods of concept generation, some well-accepted approaches that do not require a set of interacting designers also exist. Lateral thinking techniques help stimulate creative solutions using mental exercises to help encourage broad, sideways thought during the early stages of conceptual development (de Bono, 1970). Designing by analogy is another established approach to arrive at novel design solutions. This method begins by first generalizing the design problem to a set of functional requirements (or a function structure representation). Then, the functional framework allows a designer to look for or conceive of analogous products or components that perform the same set of functions (McAdams and Wood, 2000; Linsey, *et al.*, 2005). Function-means trees and morphological analysis (Zwicky, 1969) are similar methods in which solutions to individual functional requirements are first sought and then synthesized together. Apart from these approaches, one widely used method is the Theory of Inventive Problem Solving (also known by the Russian-based acronym TRIZ, (Altshuller, 1984)). This method provides a tabulated representation of a large number of solution principles that have been extracted from existing patents. Another approach is “catalog design” where concepts are generated purely through browsing a catalog of physical elements (components, assemblies, etc.). The results are evidently limited by the breadth of the catalog; however, the benefit lies in the presentation of design knowledge that falls outside the designer’s expertise memory (McAdams and Wood, 2000). The computational concept generation methods presented in this dissertation exploit the benefits of having a catalogue of design knowledge from which to pull new designs while leveraging functional descriptions to quickly home in on solutions that are the most relevant to the design problem at hand.

2.5.2. Automated Design Tools. Innovations cited by Antonsson and Cagan (2001) indicate that certain parts of larger design problems can be solved automatically and without human expertise. However, automation in the design process is often only employed once basic design concepts have been selected but lack specific dimensions. Complete automation of the design process seems to be restricted by a lack of continuity between conceptual design methods and computational design tools. Several existing design tools primarily focus on the initial design phases, such as customer need gathering, the mapping of requirements to functionality, or function decomposition (e.g. Prasad, 1998; Feng, *et al.*, 2001; Kitamura, *et al.*, 2004). Other tools address automation issues during the later steps of design embodiment or detail design, such as predicting performance in early physical embodiment designs, analyzing kinematic designs, predicting required assembly sequences for an early embodiment design, and defining the detailed geometry and layout for a conceptual solution (e.g. Onyebueke, *et al.*, 1995; Simpson, *et al.*, 1995; Fox, 1994; Johnson, 1998; Zha, *et al.*, 2001; Gorti and Sriram, 1996; Homem de Mello and Sanderson, 1991; Ishii, *et al.*, 1988; Thornton and Johnson, 1996). However, relatively few computational tools exist to assist designers during the conceptual phase of design, where requirements must be translated into a broad array of potential solutions that must then be roughly evaluated for predicted performance and cost.

Some tools or approaches do directly address the generation of design solutions from existing design knowledge, but are narrow in their application domain (e.g. Yates and Beaman, 1995; Hayes, 1995; Finkelstein, 1998) or exist only as limited research prototypes. For example, graph grammars have tackled specific component synthesis problems based on a desired behavior or performance (Schmidt and Cagan, 1995, 1997; Campbell, *et al.*, 1999, 2000, 2003; Kurtoglu and Campbell, 2005). Similarly, catalog design efforts can synthesize very specific products from a candidate set of components based purely on quantitative performance input/output requirements (Ward, 1989; Ward and Seering, 1993). Several technologies have utilized bond graphs to aid the translation from requirements to embodiment (e.g. Welch and Dixon, 1991; Gui and Mäntylä, 1994; Bradley, *et al.*, 1993; Oh, *et al.*, 1996; Bracewell and Sharpe, 1996; Sieger and Salmi, 1997), but this approach limits concept generation to dynamic systems for which bond graph relationships can be defined. Group technologies evolved as coding schemes that can be used to tag and recall components within a catalog or inventory of parts (Jordan, *et al.*, 2005; Opitz, 1970; Opitz, *et al.*, 1970; Opitz and Wiendahl, 1971; Girdhar and Mital, 2001a, 2001b; Shah and Bhatnagar, 1989; Henderson and Musti, 1988; Bhadra and Fischer, 1988). Some research outlines interesting methods for a designer to move from functional requirements to conceptual solutions (e.g. Umeda, *et al.*, 1996; Ulrich and Seering, 1988; Mann, 2000; Malmqvist and Svensson, 1999; Chakrabarti and Bligh, 2001) but employ little or no automation to assist a designer through the described activities. Others employ automation (e.g. Lu and Russomanno, 1999; Deng, 2002), but have steep learning curves or utilize knowledge in a way that is not easily generalizable to accommodate alternative approaches. In general, suitable computational tools that support the fuzzy leading edge of the conceptual phase are still relatively young and underdeveloped if they exist at all. Regardless of the specific concept generation methodology, all approaches begin by formulating the overall product function and breaking it into small easily solved subfunctions. Solutions to the subfunctions are sought, and the form of the device then follows from the assembly of all subfunction solutions.

From a perspective different than the functional modeling approach discussed above, a number of research efforts have sought to establish a generic computational scheme for electromechanical design. While these methods have yet to capture function on the same level understood by human designers, such approaches have been used in attempts to synthesize new electromechanical configurations. These methods use a variety of computer techniques including case-based reasoning (Navinchandra, *et al.*, 1991), constraint programming (Subramanian and Wang, 1995), qualitative symbolic algebra (Williams, 1990), or geometric algebras (Palmer and Shapiro, 1993). One of the most historically significant of these includes several approaches applying expert system formulations to specific design problems such as the paper roller system established by Mittal, *et al.* (1985).

The concept generation phase of the design process is, at best, difficult to translate into a succinct methodology that is useful to both experienced and inexperienced designers. Formalization of the conceptual design phase is an active, but relatively immature, area of research. Many formal methods of conceptual design have yet to be realized as computational algorithms. The work described in this dissertation presents an automated, mathematically based algorithm for concept generation and early concept evaluation capable of being adapted to multiple design applications. The specific focus of this research is the combination and formalization of function-based synthesis, constraint management, and design space search to create a comprehensive space of concept variants and search it for feasible design candidates.

2.6. SUMMARY

This section presented background information that supports the automated concept generation design tool proposed in this research. Details about existing theories of design and how the concept generator fits into a structured design methodology were given. Standardized vocabularies for functions and components, known as the Functional Basis and Component Basis respectively, and a web-based repository of product artifacts were presented as tools used by the concept generator to generate new concepts from existing design knowledge. Finally, an in-depth survey of literature on computational tools developed to support the design process was given.

3. AUTOMATED CONCEPT GENERATION, PART I: THEORY AND ALGORITHM

3.1. INTRODUCTION

This section introduces the proposed algorithm used to generate concept variants from a set of user-prescribed design requirements. First, the theory behind the proposed algorithm is presented in Section 3.2. Next, in Section 3.3, the procedure for transforming a set of functional requirements in the form of a functional model into a set of compatible components that comprise a complete concept variant is presented. In Section 3.4, a simple example demonstrating how the algorithm can be utilized is presented using a finite set of design components, namely a set of Tinkertoy™ construction toys.

3.2. THEORY

A theoretical challenge common to all attempts to automate the early conceptual design phase is the issue of how to convey functional relationships, or the basic purpose of a new design problem, to a computer so that it can search, retrieve and synthesize relevant design information. The theoretical approach of this research begins with a functional description of a desired product based on high level requirements from a societal need (e.g., customer needs), searches for components that solve the identified functionality, exhaustively explores all possible combinations of those components that can be physically integrated, ranks the resulting feasible concept variants based on designer specified criteria, and, finally, presents those ranked concept variants to the designer. The underpinnings of this computational theory of concept generation include the basic ideas that functionality of a product generally maps to a repeatable set of forms or components (e.g. Pahl and Beitz, 1988), that component to component connections are important identifiers for product architecture (e.g. Pimmler and Eppinger, 1994) and that abstract as well as concrete knowledge about products can be stored in a design repository (e.g. Szykman, *et al.*, 2001).

More specifically, if an existing repository of design knowledge exists which records, at a minimum, the functionality, connections, and generalized component name for each artifact of a set of known products, then that knowledge can be mined to create new products that are combinations of existing artifacts. The function-component relationships can be represented mathematically, most simply as a matrix. The same is true for connections between components. From these mathematical representations of design knowledge, all possible concept variants can be computed—an activity that, except for very simple products, is too tedious and time consuming for designers. The computations essentially mimic the key steps in function-based conceptual design: mapping function to physical solution through a mathematical form of a morphological matrix and connecting physical solutions together into feasible concept variants (including the ability to capture function- and component sharing).

The number of potential, feasible concept variants resulting from the computation can be overwhelming. Consider a morphological matrix with n subfunctions where there exist M_i, $i = 1..n$, component solutions for each subfunction. The upper bound on concept variants is combinatorial and given by:

$$CV_{\max} = \prod_{i=1}^{n} M_i \tag{3.1.}$$

Even after filtering out infeasible concept variants based on connections, a designer may be left with thousands of concept variants. This necessitates a ranking aspect to any automated concept generation approach. From a minimal set of product knowledge such as described in the previous paragraph, the frequency of component occurrence can be calculated and used as a simple measure of the confidence in the generated concept variant. Additional knowledge in the repository can allow for more sophisticated ranking of generated concept variants, for example component failure rate and types, manufacturing process, or cost. The particular approach to computing a rank will depend upon the criteria chosen.

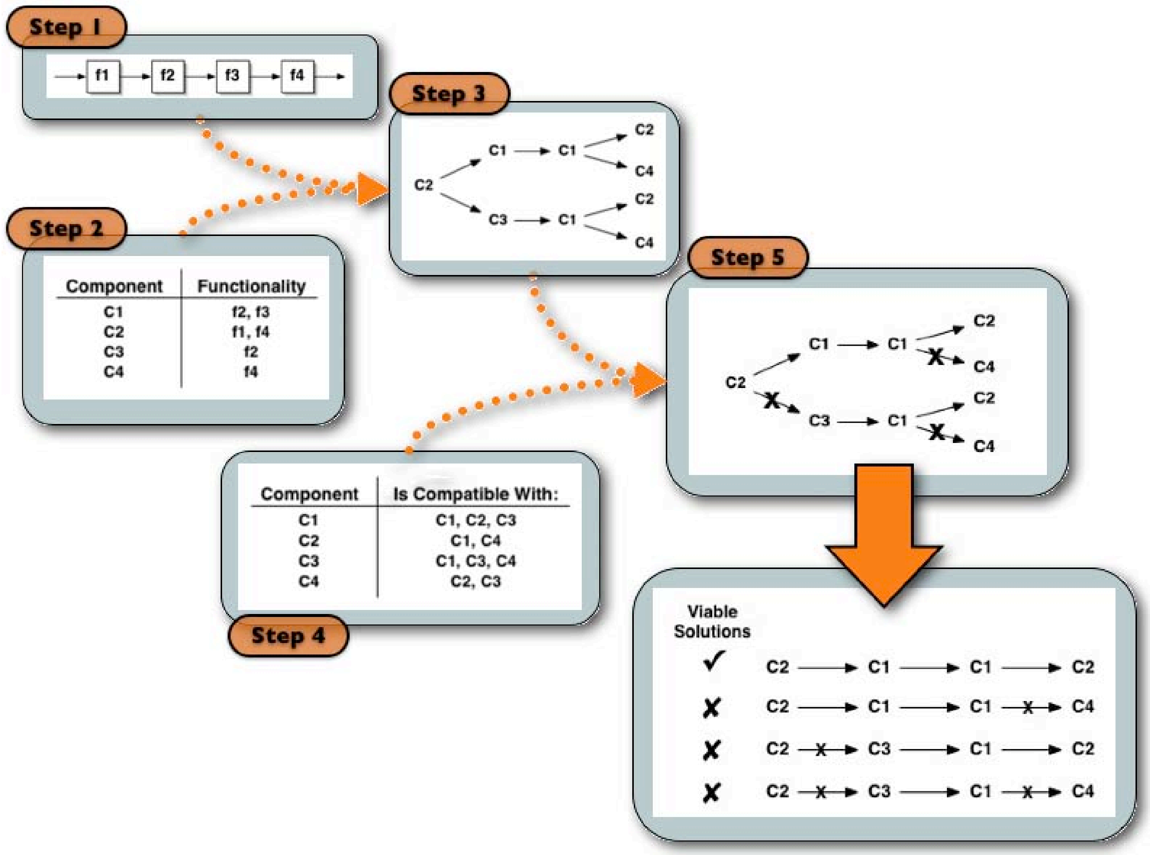
It is important to note that the *process* of the automated concept generation theory is key here. The general theory would work with any initial, abstract representation scheme to describe a product. It does not have to follow this particular form of functional modeling—in fact it does not even have to utilize function. One could potentially use customer needs as the abstract representation, for example. As long as a repository of design information, encoded by the chosen abstract representation scheme, exists, a generalized version of the mapping and computations presented would apply.

Moving to the specifics of the automated concept generation algorithm, an outline the matrix-based method of concept generation is next established. The concept generation method starts with a high level functional description of a product, expressed in the Functional Basis, and uses component functionality along with component compatibility to create, filter, and rank concept variants (Hirtz, *et al.*, 2002; Bryant, *et al.*, 2005). The function-component matrix (FCM) and the design structure matrix (DSM) describe the function-component relationships and the component-component compatibility, respectively, of existing consumer products (Pimmler and Eppinger, 1994) and are extracted from the web-based design repository hosted at the University of Missouri–Rolla. The product descriptions stored in the repository allow access to additional information such as historical occurrence and failure mode, which can be used to help limit and rank design solutions (a rudimentary ranking based on historical occurrence is implemented for results reported in this section).

3.3. ALGORITHM

The algorithm that uses the design knowledge contained in the repository to generate, filter, and rank concept variants for further analysis by design engineers is described in detail in the following sections. Figure 3.2 graphically summarizes the theory behind each step in the concept generation scheme, while Figure 3.3 relates each theoretical step it to the matrix-based manipulations necessary to compute the set of filtered concept variants.

3.3.1. Step 1: Generate a Conceptual Functional Model. The concept generation scheme begins with the functional model for either a new product to be developed or a previously developed product that is to be redesigned. Using the functional model derivation method presented in Section 2.2, a graphical block diagram that defines the flows through the product and the functions that act on those flows is created. This block diagram is then translated into a matrix form that describes the adjacency between functions, i.e., the connection between subfunctions as defined by their connecting flows. Step 1 under “Theory” in Figure 3.2 shows a simple generic flow chain of the form used to create functional models using the Functional Basis method, where f1-f4 are unspecified subfunctions of a product to be designed. Figure 3.2 also illustrates the matrix equivalent of this flow chain, an adjacency matrix where a non-zero cell entry indicates a forward connection between the row and column functions.



Component	Functionality
C1	f2, f3
C2	f1, f4
C3	f2
C4	f4

Component	Is Compatible With:
C1	C1, C2, C3
C2	C1, C4
C3	C1, C3, C4
C4	C2, C3

Figure 3.1. Visual summary of the algorithm used in the concept generator. The information shown in Steps 1, 2, and 4 is entered by the user. The unfiltered set of concept variants (Step 3) and set of feasible variants filtered by the component capability information from the DSM (Step 5) are produced using matrix algebra operations shown in Figure 3.2.

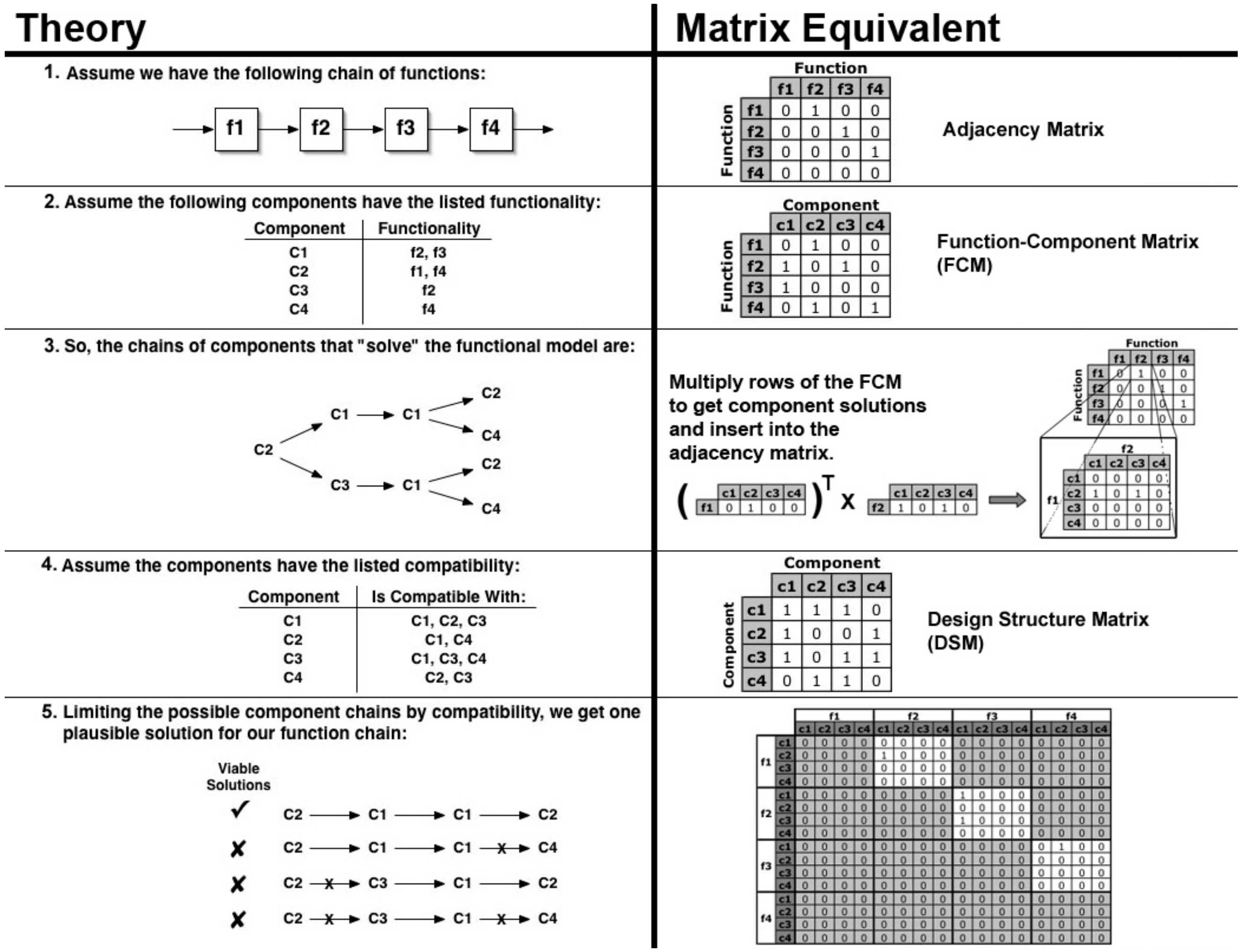

Theory
Matrix Equivalent
1. Assume we have the following chain of functions:
f1
f2
f3
f4
Adjacency Matrix
Function
2. Assume the following components have the listed functionality:
Component
Functionality
C1 f2, f3
C2 f1, f4
C3 f2
C4 f4
Function-Component Matrix (FCM)
3. So, the chains of components that "solve" the functional model are:
C2
C1
C3
C4
Multiply rows of the FCM to get component solutions and insert into the adjacency matrix.
4. Assume the components have the listed compatibility:
Is Compatible With:
C1 C1, C2, C3
C2 C1, C4
C3 C1, C3, C4
C4 C2, C3
Design Structure Matrix (DSM)
5. Limiting the possible component chains by compatibility, we get one plausible solution for our function chain:
Viable Solutions
✓ C2 → C1 → C1 → C2
✗ C2 → C1 → C1 → C4
✗ C2 → C3 → C1 → C2
✗ C2 → C3 → C1 → C4

Figure 3.2. Summary of the matrix operations for the concept generation algorithm.

3.3.2. Step 2: Define Function-Component Relationships. The next step utilizes design knowledge gathered from existing consumer products to define the relationships between a component and the functions that it solves in previously examined products. Reverse engineering techniques are applied to existing consumer products, and information extracted from each product's bill of materials and functional model is stored in the web-based design repository described in Section 2.4. The function-component relationships in the repository capture both function- and component sharing cases. In the case of function sharing, a single artifact in the repository can be tagged with as many functions as it solves. For component sharing, where several distinct components are required to solve an overall function, the components are grouped as an assembly and treated as a single artifact. Function-component matrices (FCM) for individual products or specified groups of products can easily be generated from the stored information. Non-zero cell entries in the FCM indicate that the component from the column containing the cell can solve the function from the row containing the cell. Step 2 in Figure 3.2 shows how the FCM equivalent describes the function-component relationships in the generic example shown under the "Theory" column.

3.3.3. Step 3: Compute the Set of Concept Variants that Solve the Function Model. Step 3 utilizes the information from Step 1 and Step 2 to create a set of design solutions. In Step 3 under "Theory" in Figure 3.2, a component "tree" is created showing the chains of components that could potentially solve the flow chain presented in Step 1, based on the component-function relationship information shown in Step 2. Although the generic example illustrated in Figure 3.2 shows a single branching tree (i.e., only one component solves the first function in the chain) for Step 3, it is important to note that multiple branching trees may be formed at this stage when multiple components have the potential to solve the initiating function in the chain. It is also important to note that the algorithm supports the cases of function sharing (note the repeated components C1 in the top half of the branch in Step 3 of Figure 3.2) and component sharing (here an assembly of components that solves a single function or an overall higher level function can be entered as an artifact in the repository).

Computationally, if the transpose of the row vector from the FCM that corresponds to each of the functions from the flow chain in Step 1 is matrix multiplied by the row vector from the FCM that corresponds to the forward connected function, a component-component matrix will be generated for each function connection in the flow chain. This matrix multiplication is illustrated as the matrix equivalent to Step 3 in Figure 3.2. Non-zero cells within these newly created component-component matrices represent all theoretically possible component combinations that will solve each pairing of connected functions in the flow chain.

If these component-component matrices are then placed into the adjacency matrix generated in Step 1, component paths can be traced through the aggregated matrix similar to the way a path is traced along the tree shown in Step 3 under “Theory” in Figure 3.2. Tracing every possible “path” of connections will give a list of all theoretically possible component chain variations that solve the function chain presented in Step 1 of Figure 3.2.

3.3.4. Step 4: Define Component-Component Compatibility Using Existing Design Knowledge. The next step uses additional design knowledge gathered from existing consumer products to define the compatibility between components in the examined products. As each product is reverse engineered, information regarding the connection between components is extracted from assembly models (Rajagopalan, *et al.*, 2005) and stored in the web-based design repository described in Section 2.4. Component-component compatibility matrices for individual or specified groups of products can easily be generated from the stored information. Non-zero cell entries in the component-component matrix (frequently called a design-structure matrix or DSM) indicate that the component from the column containing the cell has been directly connected to the component from the row containing the cell in an existing product. Step 4 in Figure 3.2 shows the DSM equivalent describing the known component-component compatibility in the generic example shown under the "Theory" column.

3.3.5. Step 5: Filter Set of Conceptual Variants. Step 5 uses the component compatibility information contained in the DSM to prune the tree of design solutions computed in Step 3. Shown under the "Theory" column for Step 5 in Figure 3.2, each component connection in each component chain is checked for known compatibility using the stored connection information from Step 4. An 'X' indicates each component connection line that is not supported by the compatibility table shown in Step 4. In the matrix equivalent, each cell of the DSM is multiplied with the corresponding cell in each of the function pair component-component matrices generated in Step 3. Overlaying the DSM on each matrix created in Step 3 (via cell multiplication) has the effect of removing any of the possible component connections that do not appear in the repository database. This technique uses the “experience” contained in the repository to filter out potentially inadequate concept variants and reduce the set of possible concept variants down to a more manageable size. After the matrices are filtered, we can once again trace every “path” of possible components to generate a list of feasible component chains that solve the function chain from Step 1.

Finally, this filtered list of feasible solutions can be ranked to bubble the most promising solutions to the top of the list based on a designer’s specified needs. For instance, various measures of design needs (e.g. manufacturability, recyclability, failure etc.) entered as the non-zero FCM and/or DSM entries can be used to rank and sort the resulting conceptual design solutions generated by this method. Once the set of filtered concept variants has been computed and ranked, a designer is then free to sift through the generated concept variants and evaluate the application of each to the design situation at hand.

The presented algorithm illustrates a method to quickly produce and sort a set of conceptual designs for a new or redesigned product. Functions comprising a proposed product’s functional model are mapped to lists of components that are capable of solving each function. The tree of possible component chains is then pruned by eliminating infeasible component connections according to historical component-component compatibility. This filtered set of component chains is then ranked and presented to the design engineer for further analysis. The following section illustrates the application of the presented algorithm by manually applying it to a simplified design example using Tinkertoy™ parts as the set of components available in a simulated design repository.

3.4. ILLUSTRATIVE EXAMPLE

A tricycle built from a standard Tinkertoy™ set, shown in Figure 3.3, is next presented as a simple proof-of-concept example. This simplified example demonstrates the effectiveness of the described methodology while utilizing a manageable set of artifacts for ease of illustration.

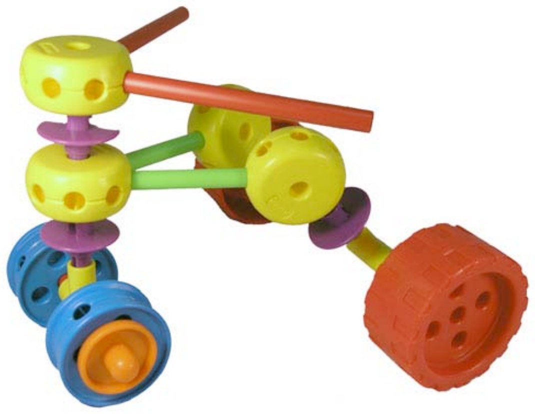
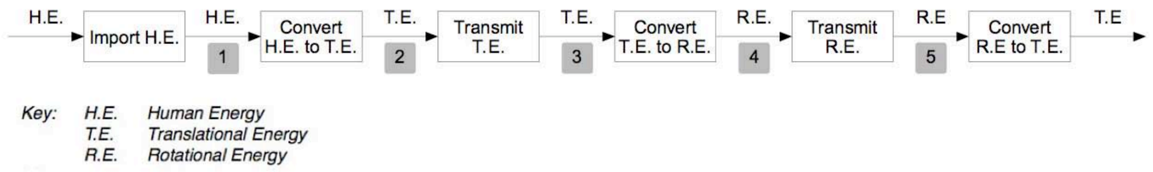


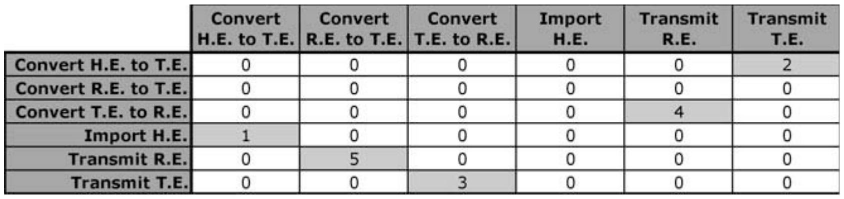
Figure 3.3. Tinkertoy™ tricycle used as the "product" to be redesigned in the following example.

First, a functional model of the tricycle construction was generated as described in Step 1 of the concept generation algorithm. For demonstration purposes, the subsequent steps of the concept generation scheme were only applied to the energy flow chain, shown in Figure 3.4a, from the complete functional model. The functional model of this flow chain begins by importing human energy across the product boundary of the tricycle toy. The model follows the energy flow as it gets converted to translational energy and transmitted through the product, then gets converted into rotational energy, which is further transmitted through the product and finally converted back into translational energy (note that we have used the tertiary categorization of flows in order to distinguish between the rotational and translation aspects of mechanical energy). Figure 3.4b shows the function adjacency matrix generated from the energy function chain in Figure 3.4a. Sequential numbers are used in the function connectivity matrix for easy reference to the connections labeled in the energy function chain.

(a)



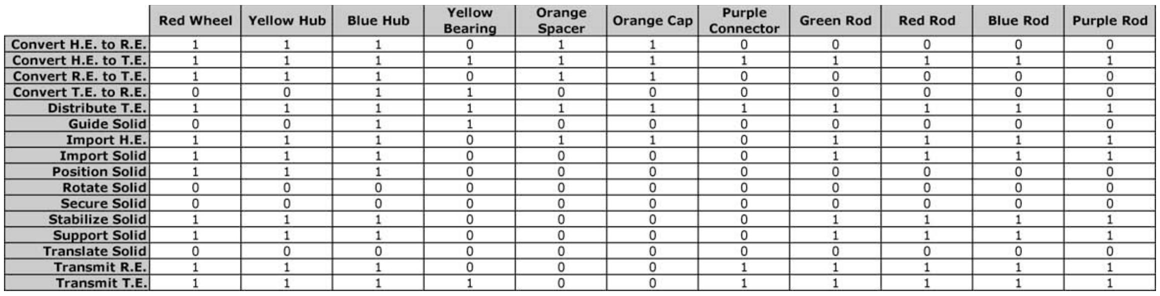
(b)



	Convert H.E. to T.E.	Convert R.E. to T.E.	Convert T.E. to R.E.	Import H.E.	Transmit R.E.	Transmit T.E.
Convert H.E. to T.E.	0	0	0	0	0	2
Convert R.E. to T.E.	0	0	0	0	0	0
Convert T.E. to R.E.	0	0	0	0	4	0
Import H.E.	1	0	0	0	0	0
Transmit R.E.	0	5	0	0	0	0
Transmit T.E.	0	0	3	0	0	0

Figure 3.4. (a) Function chain for the energy flow through the Tinkertoy™ tricycle. (b) Function adjacency matrix describing the function connections graphically shown in (a).

Next, in Step 2 of the concept generation scheme, a function-component matrix (FCM) was constructed for the complete set of Tinkertoy™ components. The FCM for the Tinkertoy™ set was generated by assigning functionality to each component of the Tinkertoy™ component set, which is, in effect, a mini-repository of Tinkertoys™. Note that the component naming terms were not used in this initial proof of concept study. The complete FCM generated for the Tinkertoy™ set is shown in Figure 3.5. For instance, the FCM indicates that the yellow bearing component is capable of embodying the following functionality: Guiding a solid, distributing translational energy, transmitting translational energy, converting human energy to translational energy, and converting translational energy to rotational energy.



	Red Wheel	Yellow Hub	Blue Hub	Yellow Bearing	Orange Spacer	Orange Cap	Purple Connector	Green Rod	Red Rod	Blue Rod	Purple Rod
Convert H.E. to R.E.	1	1	1	0	1	1	0	0	0	0	0
Convert H.E. to T.E.	1	1	1	1	1	1	1	1	1	1	1
Convert R.E. to T.E.	1	1	1	0	1	1	0	0	0	0	0
Convert T.E. to R.E.	0	0	1	1	0	0	0	0	0	0	0
Distribute T.E.	1	1	1	1	1	1	1	1	1	1	1
Guide Solid	0	0	1	1	0	0	0	0	0	0	0
Import H.E.	1	1	1	0	1	1	0	1	1	1	1
Import Solid	1	1	1	0	0	0	0	1	1	1	1
Position Solid	1	1	1	0	0	0	0	0	0	0	0
Rotate Solid	0	0	0	0	0	0	0	0	0	0	0
Secure Solid	0	0	0	0	0	0	0	0	0	0	0
Stabilize Solid	1	1	1	0	0	0	0	1	1	1	1
Support Solid	1	1	1	0	0	0	0	1	1	1	1
Translate Solid	0	0	0	0	0	0	0	0	0	0	0
Transmit R.E.	1	1	1	0	0	0	1	1	1	1	1
Transmit T.E.	1	1	1	1	0	0	1	1	1	1	1

Figure 3.5. Function-component matrix manually generated for the set of Tinkertoy™ components.

Using the function connectivity information from Figure 3.4b and the component functionality from Figure 3.5, the entire set of theoretical concept variants for the redesign was calculated during Step 3 of the concept generation algorithm. As illustrated in Figure 3.6, rows for each of the connected function pairs were multiplied together to generate the unfiltered matrices of design solutions for each function pair. These unfiltered matrices are then embedded into the function adjacency matrix to describe the full set of theoretical solutions.

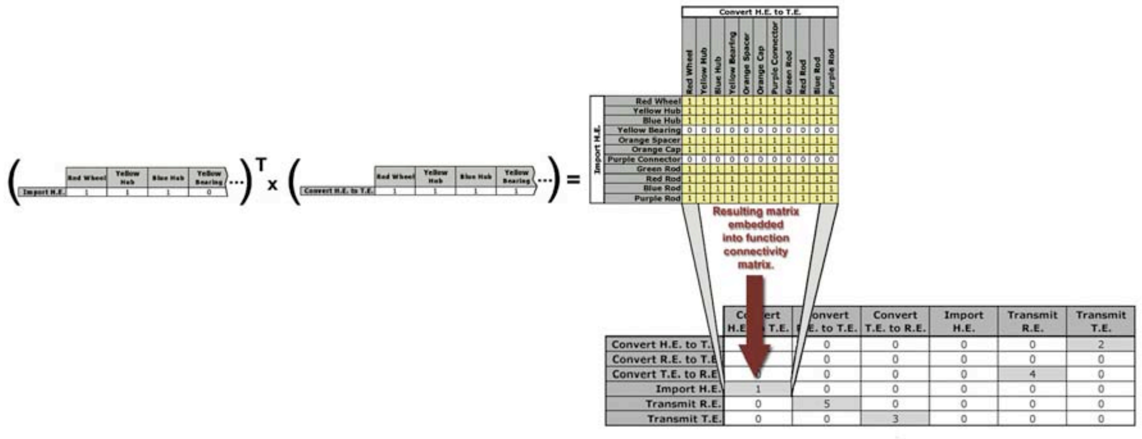
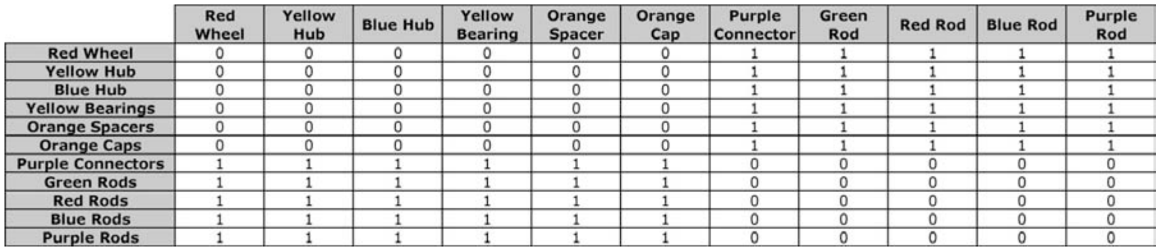


Figure 3.6. Matrix row multiplication is used to generate the set of theoretical design solutions for each connected function pair. Resulting matrices are embedded in the function adjacency matrix.

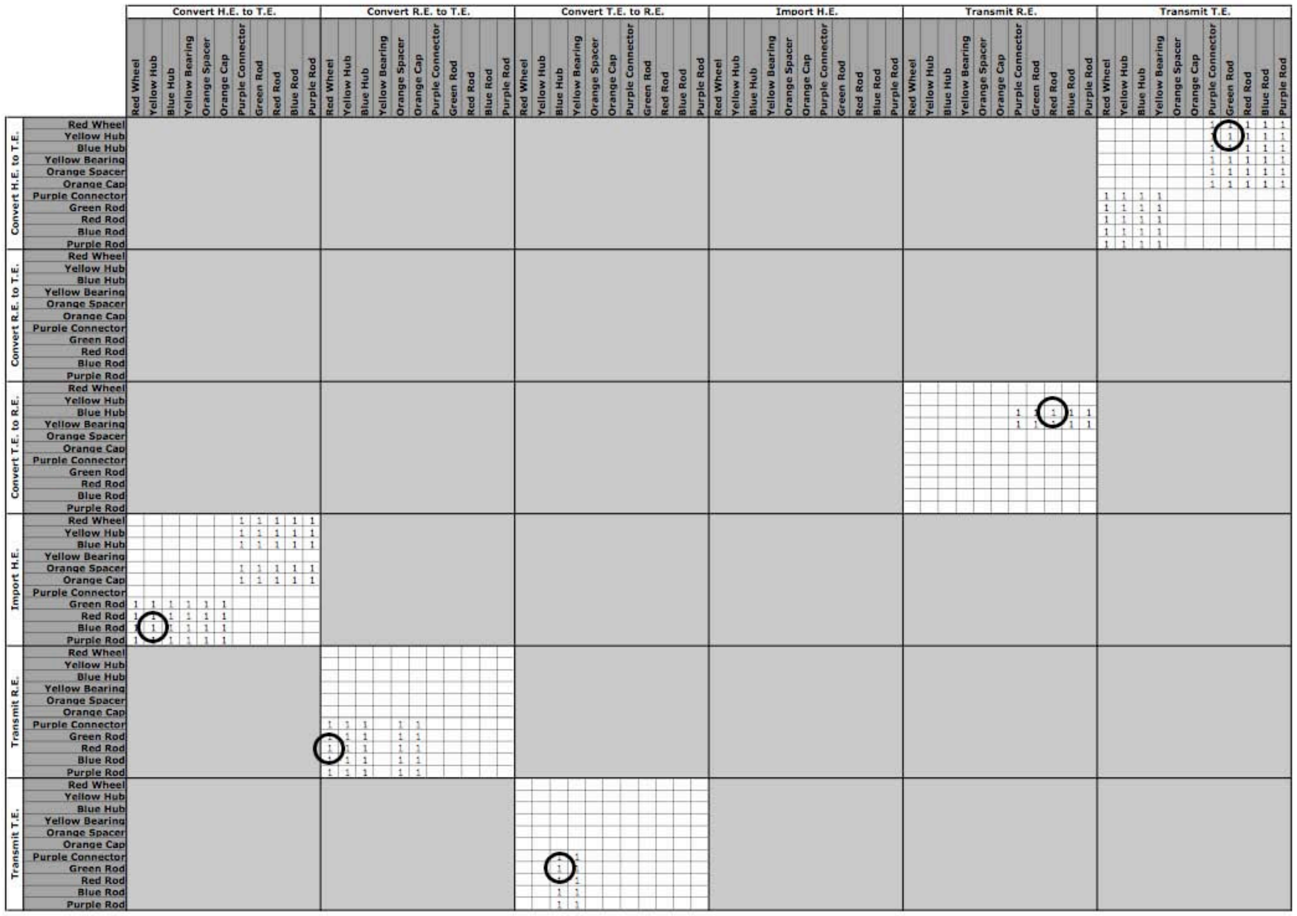
In Step 4, a similar method to that used to create the FCM was employed to construct the design structure matrix (DSM) for the set of Tinkertoy™ components. The DSM, shown in Figure 3.7, describes the component compatibility between each component, where 1’s entered into each cell identifies components that can be connected together, and 0’s indicate incompatibility.



	Red Wheel	Yellow Hub	Blue Hub	Yellow Bearing	Orange Spacer	Orange Cap	Purple Connector	Green Rod	Red Rod	Blue Rod	Purple Rod
Red Wheel	0	0	0	0	0	0	1	1	1	1	1
Yellow Hub	0	0	0	0	0	0	1	1	1	1	1
Blue Hub	0	0	0	0	0	0	1	1	1	1	1
Yellow Bearings	0	0	0	0	0	0	1	1	1	1	1
Orange Spacers	0	0	0	0	0	0	1	1	1	1	1
Orange Caps	0	0	0	0	0	0	1	1	1	1	1
Purple Connectors	1	1	1	1	1	1	0	0	0	0	0
Green Rods	1	1	1	1	1	1	0	0	0	0	0
Red Rods	1	1	1	1	1	1	0	0	0	0	0
Blue Rods	1	1	1	1	1	1	0	0	0	0	0
Purple Rods	1	1	1	1	1	1	0	0	0	0	0

Figure 3.7. Design structure matrix (DSM) generated for the set of Tinkertoy™ components.

Finally, in Step 5, each cell of the DSM was multiplied by the corresponding cell for each of the connected function pairs in order to filter out design solutions that are infeasible due to component incompatibility. The entire set of filtered design solutions is shown in Figure 3.8. To clarify the pertinent information, cells that contained zero values in the original function adjacency matrix are grayed out.



		Convert H.E. to T.E.											Convert R.E. to T.E.											Convert T.E. to R.E.											Import H.E.											Transmit R.E.											Transmit T.E.										
		Red Wheel	Yellow Hub	Blue Hub	Yellow Bearing	Orange Spacer	Orange Cap	Purple Connector	Green Rod	Red Rod	Blue Rod	Purple Rod	Red Wheel	Yellow Hub	Blue Hub	Yellow Bearing	Orange Spacer	Orange Cap	Purple Connector	Green Rod	Red Rod	Blue Rod	Purple Rod	Red Wheel	Yellow Hub	Blue Hub	Yellow Bearing	Orange Spacer	Orange Cap	Purple Connector	Green Rod	Red Rod	Blue Rod	Purple Rod	Red Wheel	Yellow Hub	Blue Hub	Yellow Bearing	Orange Spacer	Orange Cap	Purple Connector	Green Rod	Red Rod	Blue Rod	Purple Rod	Red Wheel	Yellow Hub	Blue Hub	Yellow Bearing	Orange Spacer	Orange Cap	Purple Connector	Green Rod	Red Rod	Blue Rod	Purple Rod	Red Wheel	Yellow Hub	Blue Hub	Yellow Bearing	Orange Spacer	Orange Cap	Purple Connector	Green Rod	Red Rod	Blue Rod	Purple Rod
Convert H.E. to T.E.	Red Wheel																																																																					1	1	1	1	1
	Yellow Hub																																																																					1	1	1	1	1
	Blue Hub																																																																					1	1	1	1	1
	Yellow Bearing																																																																					1	1	1	1	1
	Orange Spacer																																																																					1	1	1	1	1
	Orange Cap																																																																					1	1	1	1	1
	Purple Connector																																																							1	1	1	1							
	Green Rod																																																							1	1	1	1							
	Red Rod																																																							1	1	1	1							
	Blue Rod																																																							1	1	1	1							
	Purple Rod																																																							1	1	1	1							
Convert R.E. to T.E.	Red Wheel																																																																			
	Yellow Hub																																																																			
	Blue Hub																																																																			
	Yellow Bearing																																																																			
	Orange Spacer																																																																			
	Orange Cap																																																																			
	Purple Connector																																																																			
	Green Rod																																																																			
	Red Rod																																																																			
	Blue Rod																																																																			
	Purple Rod																																																																			
Convert T.E. to R.E.	Red Wheel																																																																			
	Yellow Hub																																																																			
	Blue Hub																																																										1	1	1	1	1											
	Yellow Bearing																																																										1	1	1	1	1											
	Orange Spacer																																																																			
	Orange Cap																																																																			
	Purple Connector																																																																			
	Green Rod																																																																			
	Red Rod																																																																			
	Blue Rod																																																																			
	Purple Rod																																																																			
Import H.E.	Red Wheel							1	1	1	1	1																																																						
	Yellow Hub							1	1	1	1	1																																																						
	Blue Hub							1	1	1	1	1																																																						
	Yellow Bearing																																																																			
	Orange Spacer							1	1	1	1	1																																																						
	Orange Cap							1	1	1	1	1																																																						
	Purple Connector																																																																			
	Green Rod	1	1	1	1	1	1																																																													
	Red Rod	1	1	1	1	1	1																																																													
	Blue Rod	1	1	1	1	1	1																																																													
	Purple Rod	1	1	1	1	1	1																																																													
Transmit R.E.	Red Wheel																																																																			
	Yellow Hub																																																																			
	Blue Hub																																																																			
	Yellow Bearing																																																																			
	Orange Spacer																																																																			
	Orange Cap																																																																			
	Purple Connector												1	1	1		1	1																																																						
	Green Rod												1	1	1		1	1																																																						
	Red Rod												1	1	1		1	1																																																						
	Blue Rod												1	1	1		1	1																																																						
	Purple Rod												1	1	1		1	1																																																						
Transmit T.E.	Red Wheel																																																																			
	Yellow Hub																																																																			
	Blue Hub																																																																			
	Yellow Bearing																																																																			
	Orange Spacer																																																																			
	Orange Cap																																																																			
	Purple Connector																								1	1																																										
	Green Rod																								1	1																																										
	Red Rod																								1	1																																										
	Blue Rod																								1	1																																										
	Purple Rod																								1	1																																										

Figure 3.8. Function adjacency matrix with embedded component connection information that describes the complete set of feasible design solutions for the tricycle redesign.

Figures 3.9a-d present four of the design variants encompassed in the matrix presented in Figure 3.8. The design variants shown are unstable asymmetric versions of the original tricycle concept since the energy function chain generated in Step 1 does not encompass requirements that the design be stable. The design variant in Figure 3.9a was constructed by selecting the component connections circled in Figure 3.8. Think of the overall matrix shown in Figure 3.8 as an adjacency matrix of embedded DSM matrices. The overall matrix has rows and columns of functions (that describe the product under study). This overall adjacency matrix captures the connectivity of the functions in the functional model.

Specifically, in Figure 3.8, enter the matrix through the row labeled ‘import h.e.’ and then read over to the cell containing the embedded matrix (the non-grey cell). Read up the column from that cell and you see the column label of ‘convert h.e. to t.e.’, the function that is connected to import h.e. Now, within the cell containing the embedded DSM matrix, follow the row labeled ‘blue rod’ (the first component of concept variant 1) across to find six cells with entries of ‘1’ in them. This means the blue rod does solve the function ‘import h.e.’ and if selected can then connect to the components ‘red wheel’ through ‘orange cap,’ as indicated by the column headings above each of the cells with a ‘1’ in them. The circle in Figure 3.8 indicates that the ‘yellow hub’ is the next component to which we will connect. Next, find the row of the overall matrix labeled ‘convert h.e. to t.e.’ (the next function in the functional model chain of Figure 3.4a) and then read over to the cell with the embedded matrix. Reading up this column identifies that the next function in the chain will be ‘transmit t.e.’ Returning to the cell, we start at the row corresponding to ‘yellow hub’ (the component chosen to connect to ‘blue rod’). The ‘yellow hub’ can connect to the components ‘purple connector’ through ‘purple rod,’ as indicated by the ‘1’ entries and, in this instance, we explore connecting to the ‘green rod.’ Now, we move on to the third function in the chain, ‘transmit t.e.’ Locate the ‘transmit t.e.’ row in the overall matrix and read over to the cell containing the embedded matrix. Reading up identifies that ‘convert t.e. to r.e.’ is the next function in the chain. Within the embedded matrix, we locate the ‘green rod’ row and see that there are two possible connections – ‘blue hub’ or ‘yellow bearing.’ In this instance, the ‘blue hub’ is selected. Continuing on shows how the remaining two components that solve the functionality specified in Step 1 for this concept variant are identified.

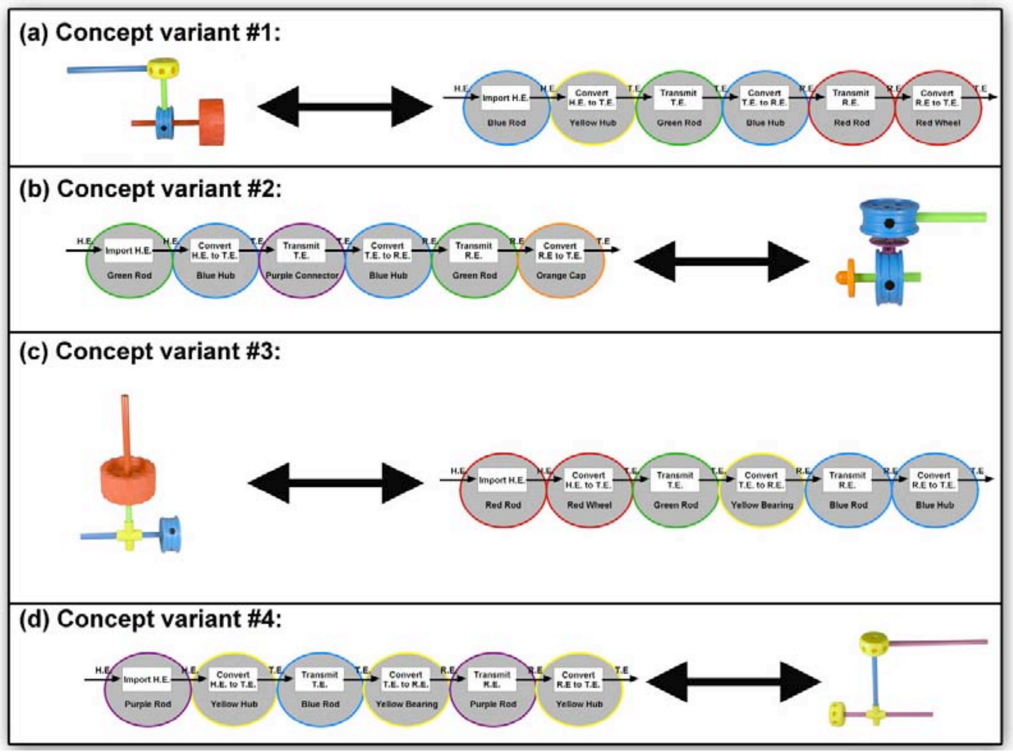


Figure 3.9. (a)-(d) Concept variants selected from the matrix of feasible solutions presented in Figure 3.8.

Using this technique, a set of feasible design solutions for the product to be designed or redesigned can be identified. Ranking of the design solutions can be accomplished by calculating a “score” for each concept variant using stored measures of frequency of occurrence, manufacturability, assemblability, or other measures related to the component connections selected. The ranking is not implemented for this contrived Tinkertoy™ example. In Section 4, the presented algorithm is automated to eliminate the need for manual matrix manipulations and quickly produce concept variants for evaluation.

4. AUTOMATED CONCEPT GENERATION, PART II: SOFTWARE

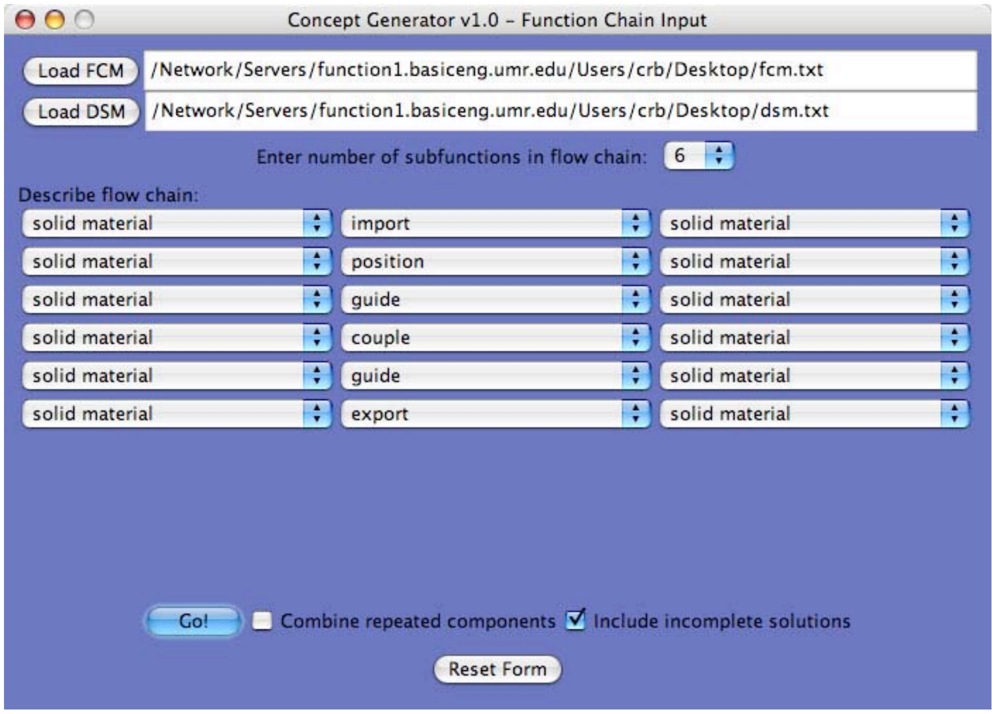
4.1. INTRODUCTION

This section presents the software implementation of the proposed algorithm presented in Section 3. The first implementation implements the computational theory presented in Section 3 and presents the designer with a list of possible component solutions that satisfies the functional requirements input by a the user. Next, Section 4.3 presents a case study that uses the list-based implementation. Finally, in Section 4.4, an improved implementation, which extends the capabilities of the software presented in Section 4.2, is presented.

4.2. AUTOMATION OF THE CONCEPT GENERATION ALGORITHM

Using the algorithm described in Section 3, a Java-based program was created to automatically produce a ranked list of concept variants for an input functional model chain. The user interface, shown in Figure 4.1, firsts prompts the user for the location of the function-component matrix (FCM) and design structure matrix (DSM) data files generated from the web-based design repository from which the new concepts will be created. Within the repository, the FCM and DSM design tools permit the user to select any subset of products from the repository from which to generate these matrices, allowing the designer to select which group of products to build new concepts from.

(a)



Concept Generator v1.0 – Function Chain Input

Load FCM: /Network/Servers/function1.basiceng.umr.edu/Users/crb/Desktop/fcm.txt

Load DSM: /Network/Servers/function1.basiceng.umr.edu/Users/crb/Desktop/dsm.txt

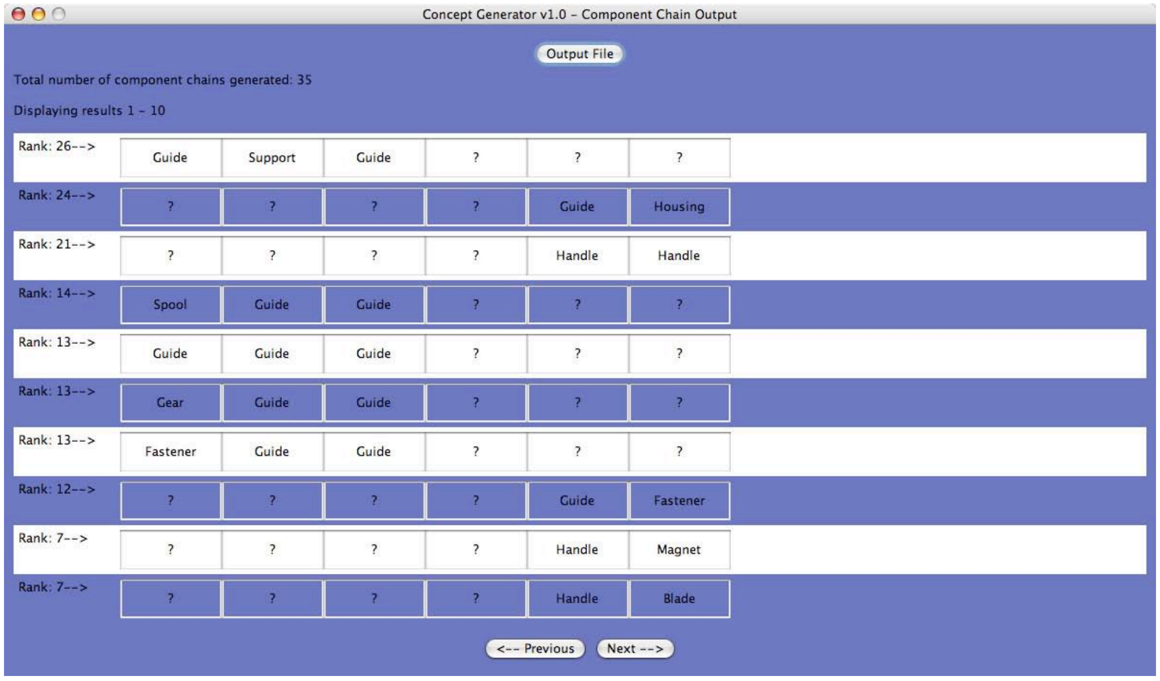
Enter number of subfunctions in flow chain: 6

Describe flow chain:

solid material	import	solid material
solid material	position	solid material
solid material	guide	solid material
solid material	couple	solid material
solid material	guide	solid material
solid material	export	solid material

☐ Combine repeated components ☑ Include incomplete solutions

(b)



Concept Generator v1.0 – Component Chain Output

Total number of component chains generated: 35

Displaying results 1 – 10

Rank: 26-->	Guide	Support	Guide	?	?	?
Rank: 24-->	?	?	?	?	Guide	Housing
Rank: 21-->	?	?	?	?	Handle	Handle
Rank: 14-->	Spool	Guide	Guide	?	?	?
Rank: 13-->	Guide	Guide	Guide	?	?	?
Rank: 13-->	Gear	Guide	Guide	?	?	?
Rank: 13-->	Fastener	Guide	Guide	?	?	?
Rank: 12-->	?	?	?	?	Guide	Fastener
Rank: 7-->	?	?	?	?	Handle	Magnet
Rank: 7-->	?	?	?	?	Handle	Blade

Figure 4.1. User interface for (a) inputting the FCM, DSM, and functional model for automatic concept generation, and (b) browsing through the list of returned concept variant chains. Actual entries correspond to the case study presented in Section 4.3.

Next, the user enters the number of distinct flow chains contained in the conceptual functional model. This initial version of the concept generation software limits flow chain entries to a single non-branching flow, requiring the user to break a full functional model up into individual chains prior to entry into the software. The user then selects the number of subfunctions in each flow chain and proceeds to enter the input and output flows and subfunctions for the individual chain. At this point, concepts can be generated and ranked for each flow chain by selecting the “Go!” button.

The number of components displayed for each concept variant can be minimized by selecting the “Combine repeated components” checkbox. Selecting this option instructs the program to search for repeating series of components in the concept variant chain and collapse them down to a single instance for display, exploiting the concept of function sharing. The option to “Include incomplete solutions” in the ranked returned concepts is also available. This allows the user to decide whether to display concept variant chains that may be incomplete (i.e. not all subfunctions have an associated component solution) since the design repository may not yet contain preexisting solutions for the entered flow/subfunction combination. If selected, incomplete variants will show a question mark in chains where a solution with known compatibility cannot be found.

After obtaining the user input, the program filters the FCM so it contains only those functions relevant to the user-input functional model. From this filtered FCM, or morphological matrix, the component-component matrices for each pairing of functions are calculated and filtered using the information contained in the DSM. The components in both the FCM and DSM are categorized according to the terms from the component naming basis presented in Section 2.3.2. Finally, all combinations of the remaining feasible component-component connections are determined, ranked, and output as potential component configurations for the input functional model. In the initial implementation of the algorithm, a rudimentary ranking of the concept variants by historical occurrence of their constituent components is calculated (note that a high ranking result indicates that the concept variant is composed of the most commonly occurring components). The magnitude of the cell values from the FCM supplies the occurrence data.

Once the concept variants are created and ranked, the results are displayed in a separate window where the user can either save the results to a text file or browse through the variants using the interface at the bottom of the panel. By using them as a point of departure for other non-computational creative techniques like brainstorming, these conceptual design variants can then be further developed and/or modified by the designer to satisfy the design requirements. The next section presents a case study for the creation of a box-labeling device to demonstrate the effectiveness of the software in a real-world design situation.

4.3. CASE STUDY: BOX LABELING DEVICE

This section presents a case study that demonstrates how the automated concept generation software can effectively assist a designer during the early phases of design. A design team was charged with creating a box-labeling device to assist workers at a local area workshop for persons with disabilities. Prior to the designer’s solution, the task of labeling the contents of cardboard boxes filled with sample products from a local business was restricted to those workers who possessed the agility and mental capacity required to properly hand write the information on the box. The managers at the workshop were looking for a solution that would allow any of the workers to perform this task regardless of level of ability, while maintaining a level of quality acceptable to the local business that contracted the work. After determining the applicable customer needs for the device to be designed, the conceptual functional model, shown in Figure 4.2, for the box-labeling device was generated.

Since the current form of the software is limited to handling single, non-branching flow chains, the functional model shown in Figure 4.2 was divided into individual non-branching chains, as illustrated in Figure 4.3. Note that subfunctions with multiple input/ output flows appear in multiple flow chains, and that these repeated subfunctions appear vertically adjacent to each other in Figure 4.3. These five flow chains were used as the input into the concept generation program as demonstrated using Flow Chain C shown in Figure 4.1a above.

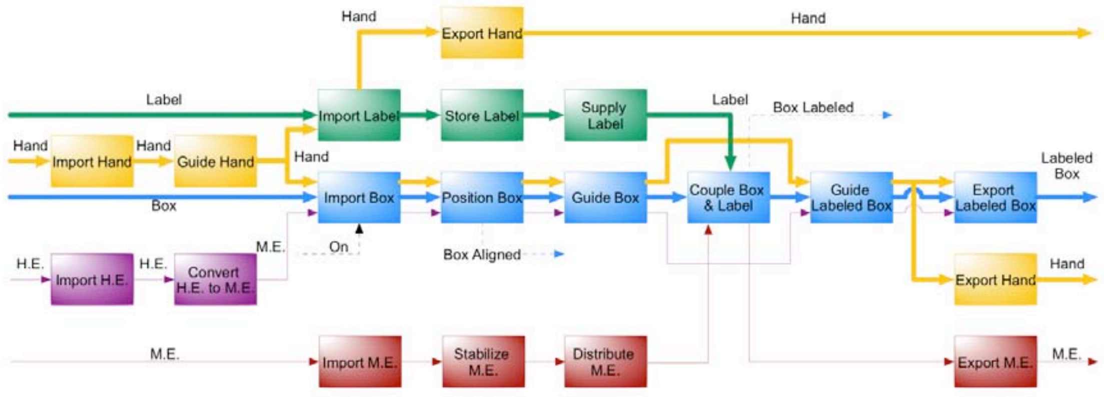


Figure 4.2. (Above) Conceptual functional model for the case study of a box-labeling device.

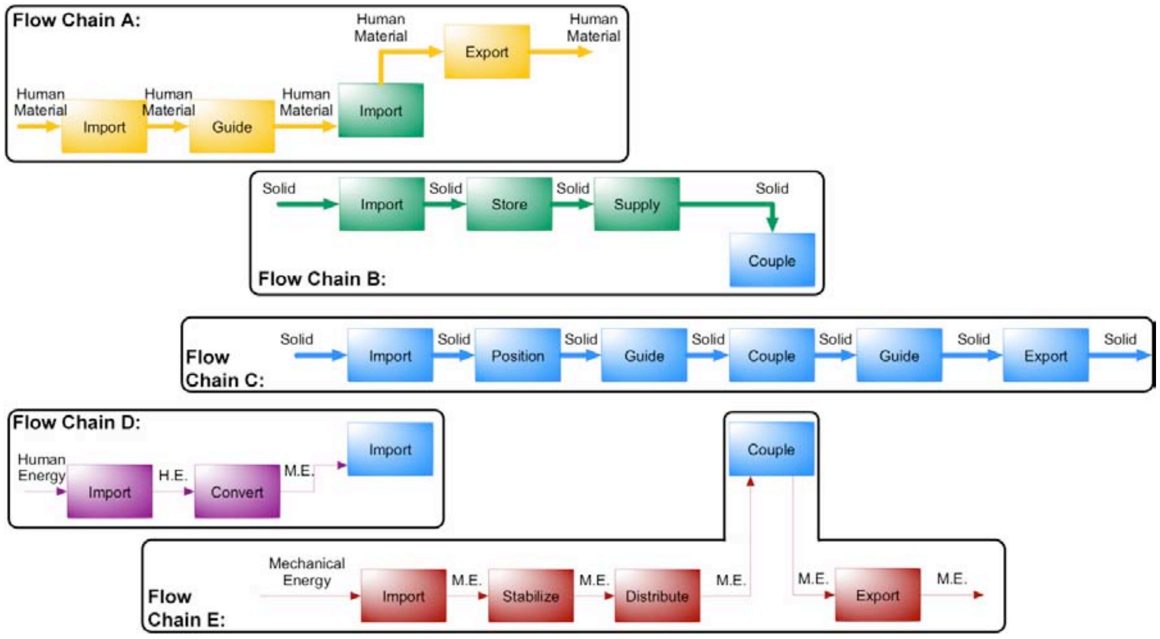
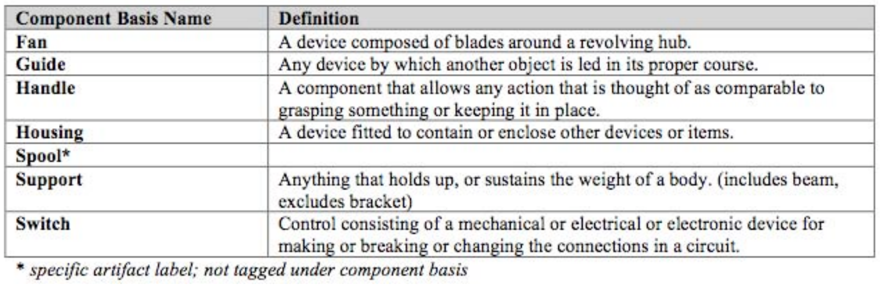


Figure 4.3. The conceptual functional model was divided into single non-branching flow chains, labeled Flow Chains A-E, and entered into the concept generation software.

The panel shown in Figure 4.1b demonstrates how the top ranked concepts are displayed for flow chain C. All returned concept variants for flow chain C can be viewed using the “Previous” and “Next” buttons located at the bottom of the panel. Additionally, the concept variant chains can be saved to a text file using the “Output File” button located at the top of the panel. Question marks located in the component placeholders for the returned variants indicate that no solutions were found with known compatibility with the adjacent component. Variants containing unknown solutions may be combined to create a more complete solution. For instance, combining the top two solutions shown in Figure 4.1b results in a concept variant with only one unspecified component solution. It is important to note that at the time that this case study was performed, the design repository contained knowledge data from a limited number of consumer products. As the repository has grown to house design information on over 100 consumer products to date, the number of incomplete solutions returned is greatly reduced. Gathered from the results returned by the algorithm, these and other top ranked component chains for each flow chain, A-E, are displayed in Figure 4.4. Vertically adjacent components designated by the dotted outlines indicate solutions for the same subfunction, which was repeated when the full functional model was dissected into individual chains. For clarity, Table 4.1 shows the definitions for the subset of Component Basis names shown in Figure 4.4.

Table 4.1. Subset of Component Basis artifacts found in the listed concept variants (Kurtoglu, *et al.*, 2005).



Component Basis Name	**Definition**
Fan	A device composed of blades around a revolving hub.
Guide	Any device by which another object is led in its proper course.
Handle	A component that allows any action that is thought of as comparable to grasping something or keeping it in place.
Housing	A device fitted to contain or enclose other devices or items.
Spool*	
Support	Anything that holds up, or sustains the weight of a body. (includes beam, excludes bracket)
Switch	Control consisting of a mechanical or electrical or electronic device for making or breaking or changing the connections in a circuit.

***** *specific artifact label; not tagged under component basis*

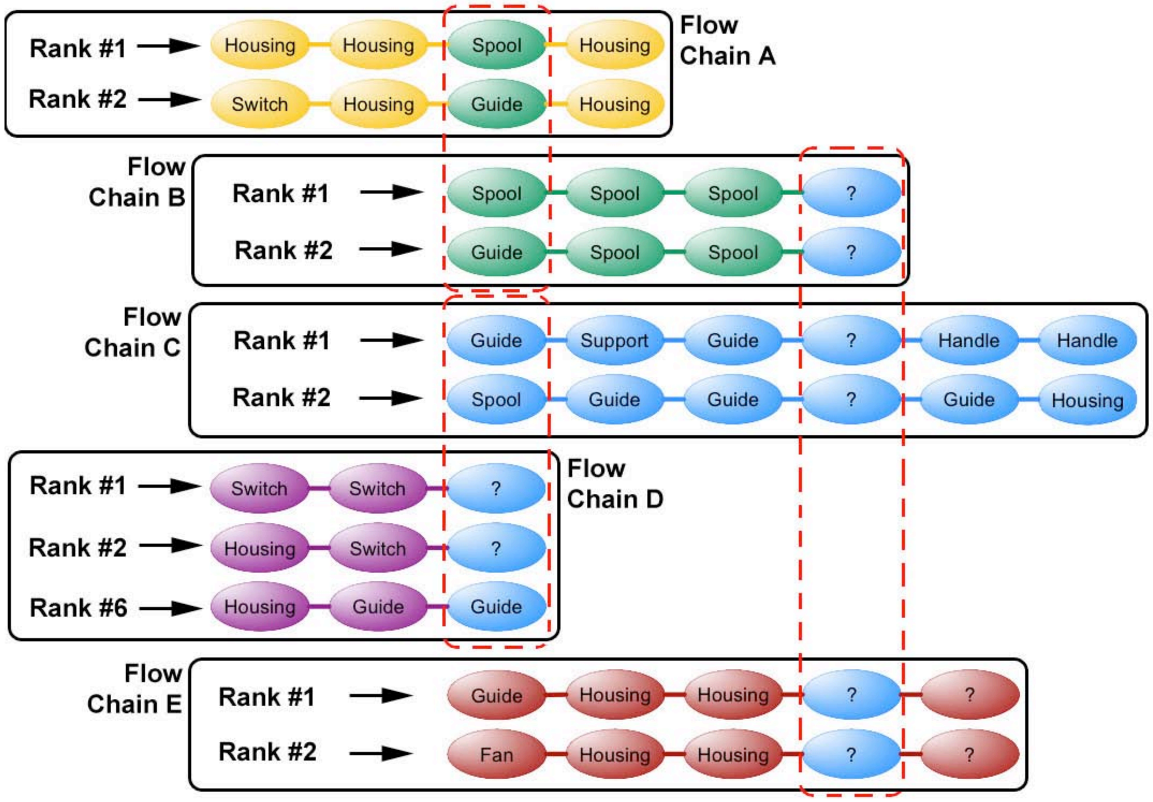


Figure 4.4. Conceptual component chains generated from the concept generation software. Components grouped together vertically by the dotted outlines indicate overlap in the component chains. This redundancy is triggered when the complete functional model is divided into individual flow chains, causing a single subfunction to appear in multiple flow chains.

The individual component chains, shown in Figure 4.4, can then be reassembled to produce a complete concept variant for the product to be redesigned (see Figure 4.5). To help clarify the component-function relationships for the concept variant chosen, the complete concept variant, shown in Figure 4.5, was superimposed onto the functional model from Figure 4.2.

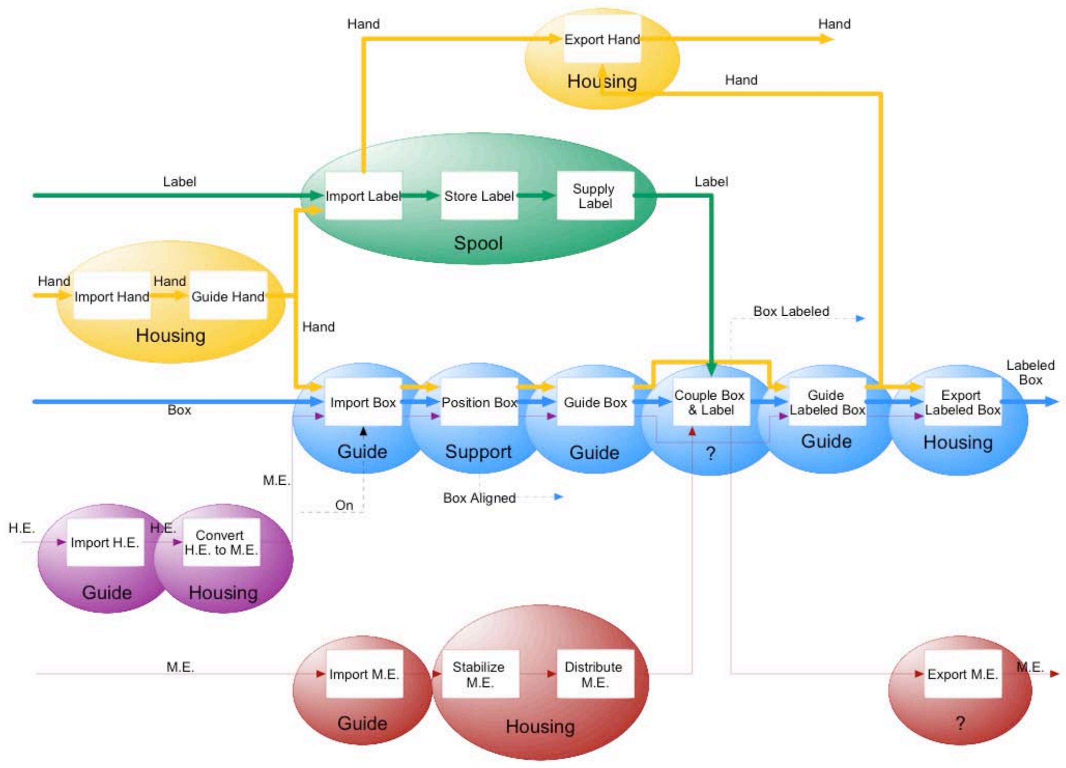


Figure 4.5. Aggregated concept variant generated from the component chains shown in Figure 4.4. Components are associated with the subfunctions from the functional model they solve.

After generating an array of concept variants from the software, sketching techniques can next be employed as a final step to create visual representations of selected conceptual design variants. Using the Component Basis definitions and pictures of specific artifacts from the web-based repository as guides, multiple embodiments of the conceptual design ideas were generated for the box-labeling device by sketching various configurations of the returned component solutions, one of which is shown in Figure 4.6.

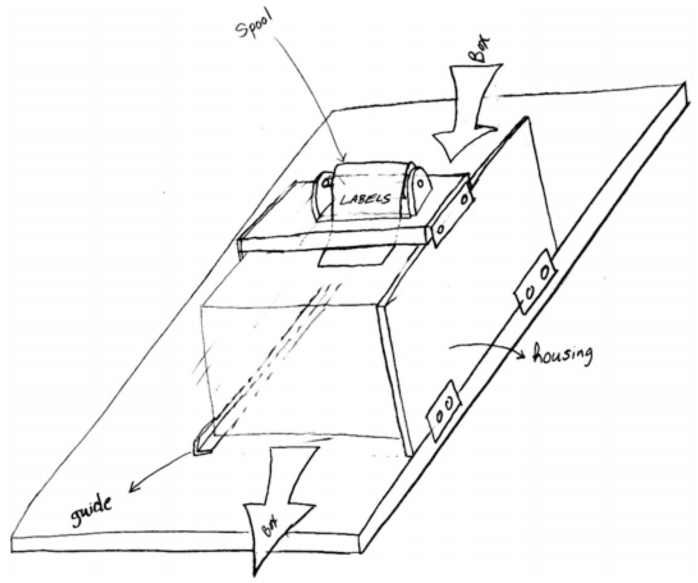


Figure 4.6. Conceptual design generated for the box-labeling device, inspired by the concept generation program output.

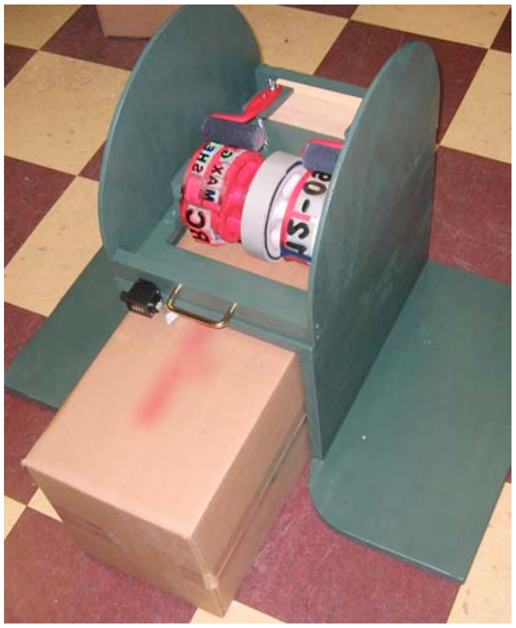


Figure 4.7. Embodied design for the box-labeling device (cover removed to show internal components).

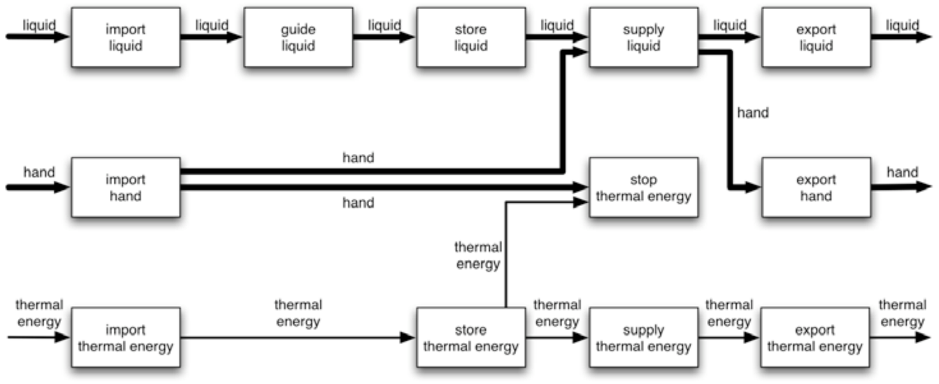
Many concepts were generated using several different methodologies over the course of the box-labeling project. After each of the concept variants generated by the various methods were evaluated and ranked, the sketch shown in Figure 4.6 was chosen as the starting point for the final box-labeling device design. Although the eventual embodiment of the box-labeling device, shown in Figure 4.7, was modified from the initial conceptual sketch presented in Figure 4.6 during the later stages of design, the concept variant shown in Figure 4.5 catalyzed the idea that led to a successful end design.

4.4. MEMIC: THE INTERACTIVE MORPHOLOGICAL SEARCH

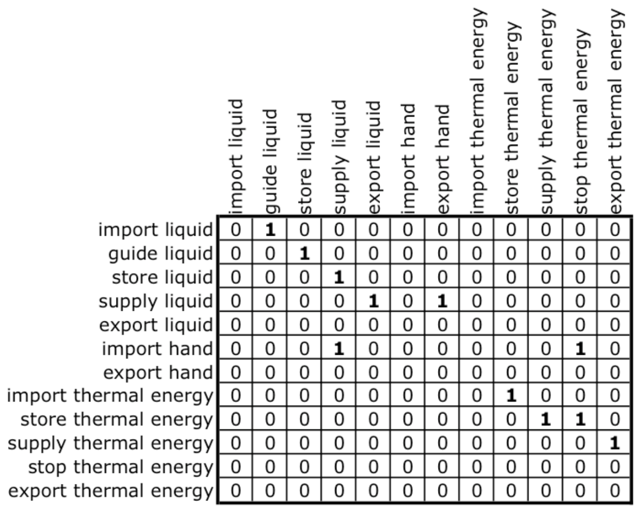
Beneficial characteristics of the web-based morphological search described in Section 2.4.4. and the automated concept generator described in Section 4.2. were combined into a hybrid technique in an effort to enhance the usefulness of the automated design tool to a designer. The hybrid technique, named MEMIC or Morphological Evaluation Machine and Interactive Conceptualizer, retains the solution accessibility that the web-based morphological search method provides a user by listing the solutions for each function in a matrix form, while retaining the connectivity information that the list-based automated concept generator establishes. Thus a user can more easily choose between multiple solutions for a given function and interactively build a complete feasible solution. The code for the MEMIC software can be found in Appendix A.

The interactive morphological search begins by accepting a text file describing a full functional model in the form of a function-adjacency matrix. A function-adjacency matrix, briefly demonstrated using a simple example shown in Figure 4.8, is a translation of a block functional diagram into matrix form, where a non-zero cell entry indicates a forward connection between the row and column functions. Converting a graphical functional model into this asymmetric matrix form yields an easy and convenient tool for identifying the connectivity between functions, including branching connections and connections that converge into a single function, as well as starting and ending subfunctions (zero columns and zero rows, respectively).

(a)



(b)



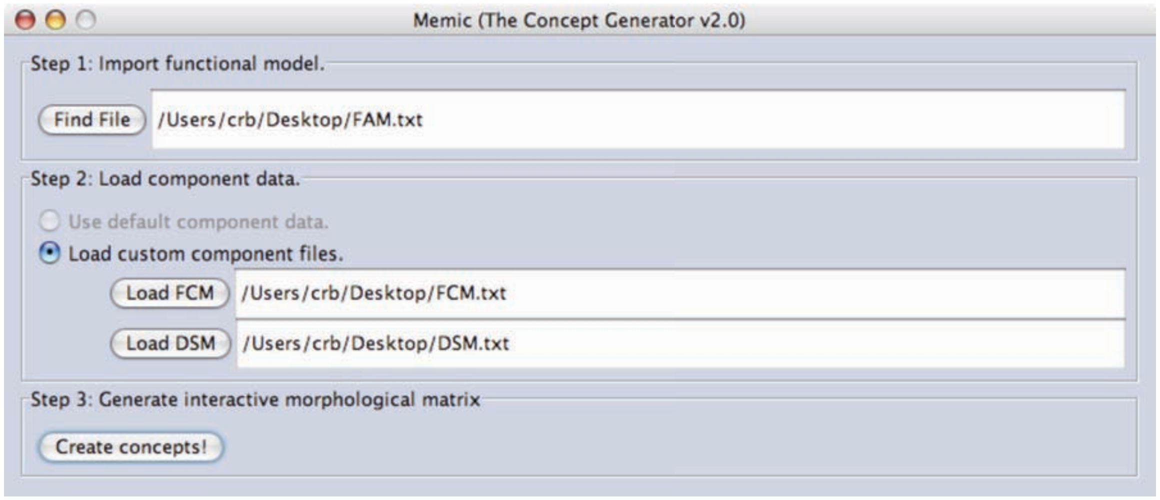
	import liquid	guide liquid	store liquid	supply liquid	export liquid	import hand	export hand	import thermal energy	store thermal energy	supply thermal energy	stop thermal energy	export thermal energy
import liquid	0	**1**	0	0	0	0	0	0	0	0	0	0
guide liquid	0	0	**1**	0	0	0	0	0	0	0	0	0
store liquid	0	0	0	**1**	0	0	0	0	0	0	0	0
supply liquid	0	0	0	0	**1**	0	**1**	0	0	0	0	0
export liquid	0	0	0	0	0	0	0	0	0	0	0	0
import hand	0	0	0	**1**	0	0	0	0	0	0	**1**	0
export hand	0	0	0	0	0	0	0	0	0	0	0	0
import thermal energy	0	0	0	0	0	0	0	0	**1**	0	0	0
store thermal energy	0	0	0	0	0	0	0	0	0	**1**	**1**	0
supply thermal energy	0	0	0	0	0	0	0	0	0	0	0	**1**
stop thermal energy	0	0	0	0	0	0	0	0	0	0	0	0
export thermal energy	0	0	0	0	0	0	0	0	0	0	0	0

Figure 4.8. (a) A simple functional model and (b) the associated function-adjacency matrix.

Next, as for the list-based concept generator, a user is prompted to load tab-delimited data files of the function-component matrix (FCM) and design structure matrix (DSM) generated from the web-based design repository. The user interface for uploading the files is shown in Figure 4.9a. Once each of these three data files is loaded, the “Get concepts” button may be pressed to generate design solutions.

When the user indicates that concepts should be generated, the data files are run through an algorithm similar to the one described in Section 3. However, to build up full solutions more efficiently and eliminate occasional solution “dead ends” that may not be weeded out using the automated concept generator in Section 4.2., the algorithm is expanded to check for and remove “dead end” solutions. The solutions are then returned to the user in the form of a morphological matrix, where the components that may be assembled into a full solution are listed alongside the name of each subfunction in the input functional model. If no compatible solution was found for a given subfunction, a “?” is placed as an indicator that no known solutions were found within the database that was also compatible with the solutions connecting to it, indicating to a designer that a novel partial solution may need to be implemented to create a complete design.

(a)



Memic (The Concept Generator v2.0)

Step 1: Import functional model.

Find File /Users/crb/Desktop/FAM.txt

Step 2: Load component data.

Use default component data. [✓ Load custom component files.]

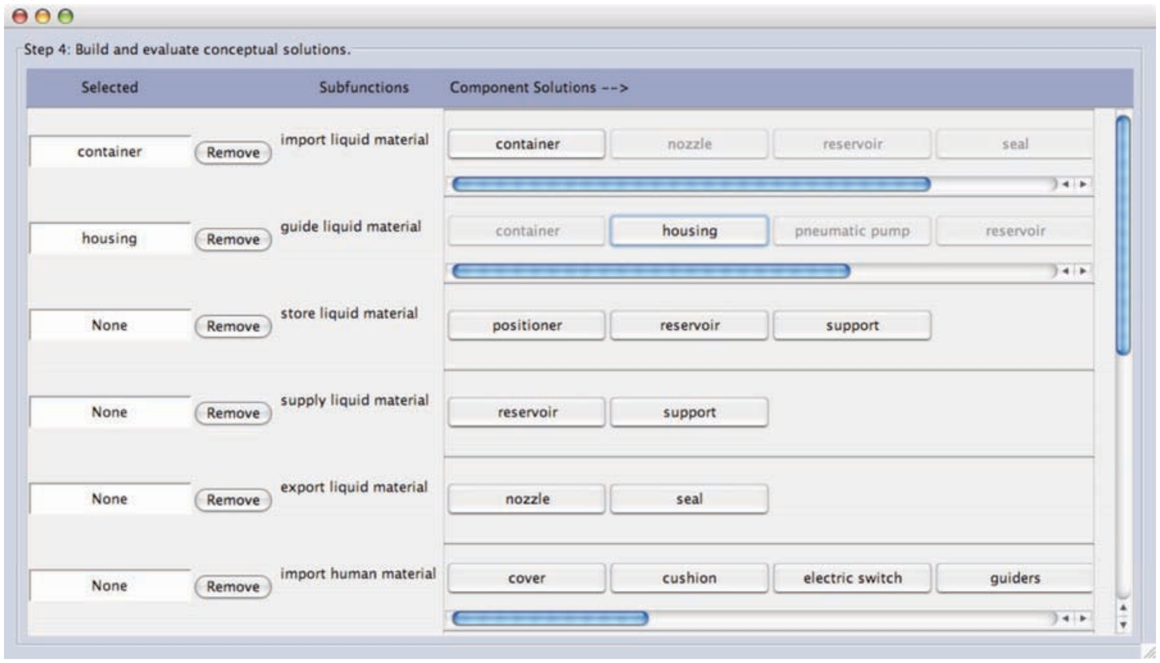
Load FCM /Users/crb/Desktop/FCM.txt

Load DSM /Users/crb/Desktop/DSM.txt

Step 3: Generate interactive morphological matrix

Create concepts!

(b)



Step 4: Build and evaluate conceptual solutions.

Selected		Subfunctions	Component Solutions -->			
container	Remove	import liquid material	container	nozzle	reservoir	seal
housing	Remove	guide liquid material	container	housing	pneumatic pump	reservoir
None	Remove	store liquid material	positioner	reservoir	support	
None	Remove	supply liquid material	reservoir	support		
None	Remove	export liquid material	nozzle	seal		
None	Remove	import human material	cover	cushion	electric switch	guiders

Figure 4.9. The interactive morphological search user interface for (a) inputting the FCM, DSM, and functional model for automatic concept generation, and (b) interacting with the return conceptual solutions.

Once components are returned, the interactive morphological matrix, shown in Figure 4.9b, allows the user to select components that solve each function in an input functional model. When a solution component is selected, incompatible solutions are shaded over and the user is no longer allowed to select them. By implementing the concept generator output in this fashion, users can build entire concepts that, based on historical data contained within the repository, are comprised only of components that can physically be connected together. By using the interactive morphological matrix, a designer is allowed to tinker with various ideas and virtually assemble a complete solution that can be physically produced.

To be effective, the component terms presented to a designer via the interactive morphological matrix must be meaningful and rigorously defined. To this end, the Component Basis described in Section 2.3.2. has been enhanced by establishing a hierarchical method to classify components and establish new classification terms. This research is present next in Section 5.

5. COMPONENT CLASSIFICATION FOR KNOWLEDGE RETRIEVAL

5.1. INTRODUCTION

To facilitate the interpretation of results presented via the interactive morphological matrix described in Section 4.4, a methodology for the systematic placement of components into a hierarchical ontology is presented. Cues taken from the Linnaean classification system for living organisms are used to generate a hierarchical ontology for organizing component terms and to create a robust procedure for adding new component terms to an existing component naming scheme. The objective of this research is to begin constructing a hierarchical ontology that is analogous to the Linnaean classification system with specific rules that rigorously guide component placement within the framework. The primary motivation for this research is to develop an ontology of distinct abstract components terms that supports computational strategies for automated design synthesis, general design knowledge storage and reuse, efficient communication of design information, and standardization for digital component cataloging and searching.

5.2. MOTIVATIONS

Components are the fundamental artifacts from which physical devices are built. In the early stages of design, a designer must take a set of specifications and constraints and translate these design requirements into a set of compatible components that work together to solve a desired task. As an electromechanical design evolves from a loose conceptual sketch to a fully realized product design, designers make decisions regarding specific component geometry and performance. While formal component representations exist during the detailed stages of product development, electromechanical components lack similar representations that support the conceptual phase of design, leaving a designer to rely on personal experience or potentially time consuming search methods to identify an initial broad selection of distinct conceptual component configurations for a design. In addition, less experienced designers may find it difficult to produce a broad array of distinctly different potential solutions, and instead may generate several similar alternatives that may contain one or more components that are merely variations on a theme within the realm of his or her personal experience. In the early stages of design, specific details of component geometry and performance are less important than the ability to represent component knowledge at a higher level of abstraction (Kuziak, *et al.*, 1991). The functionality of components provides a natural framework upon which such abstractions can be built. Previous work sought to develop and later refine a component naming convention for abstract functionally relevant component classes for first mechanical and later electromechanical components (Greer, *et al.*, 2003; Kurtoglu, *et al.*, 2005). The research presented here seeks to create a hierarchical ontology into which both new and existing component terms may be classified. It is hoped that this hierarchy, inspired by the animal classification system begun by Carolus Linneaus, will help ensure that the goal of complete and exclusive inclusion of all components into the ontology will be maintained as new terms are added.

5.2.1. Implementation of a Computational Theory for Design Synthesis. Many researchers have explored automated design tools to improve design synthesis activities (see Section 2.5.). Components typically constitute the fundamental building blocks of these activities. Within the variety of computer aided design research, various methodologies and tools have been developed which require a rich library of components, however, there is no agreed upon standard component library. As a result of this, libraries of components are independently developed in an application specific manner. Creation of a structured framework for the classification of new and existing components will reconcile previous efforts into a single electromechanical component library that can be leveraged by a number of design automation methods.

5.2.2. Design Knowledge Reuse. Over the past few decades, systematic approaches to conceptual design have emerged (see Section 2.2.). These design methods begin by formulating the product function as a set of low level subfunctions, solutions to which are then synthesized together to arrive at a final design. The core of the computational synthesis methods, presented in Sections 3 and 4, that are built upon this function-based framework is the mapping of subfunctions to components. This allows designers to generate concept variants from a generic functional description of the product being designed. Each of these computational methods requires a knowledge base of “reconfigurable” standardized component objects that can be archived, searched and reused. A defined ontology facilitates the organization of such a knowledge base so that various computational design tools can leverage existing design knowledge.

5.2.3. Communication of Design Knowledge. The use of natural language often leads to ambiguity in representing component design knowledge. Arbitrary and redundant component naming results in different interpretations among designers for similar concepts, hindering effective communication of design knowledge. By associating fundamental component concepts with uniquely defined component classes and by providing a structure for defining each term, improvements in uniformity and consistency in the representation of components and communication of design information for industry and design education are possible.

5.2.4. Standardization for Digital Component Cataloging. Solutions to conceptual design problems are usually represented as a configuration of components and interactions between them (Kurtoglu and Campbell, 2005; Liang and Paredis, 2004). The transformation from these configurations to fully embodied design solutions requires the specification of a system of electromechanical components that meet the overall design requirements. 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. As a result, the OEM suppliers compete by striving to improve their components quality and variety. It is particularly important for them to catalogue their solutions such that they can be efficiently retrieved and incorporated into the design process. Technologies involving electronic representations of standard components and resulting digital databases are becoming more prominent in engineering design (Wallace, 1995; Culley and Webber, 1992; Hicks, *et al.*, 2005). Contributing to these efforts, it is hoped that this ontology provides a useful classification scheme for vendors selling a variety of OEM components.

Motivated by these factors, a starting point for the creation of a component ontology that is accessible to all design engineers is provided here. In the following subsections, other approaches to cataloging components, the use of ontologies in engineering design and computational synthesis, and a discussion of the biological parallels between classifying animals and classifying components will be discussed. The background discussions are followed by a description of the method used to create the proposed hierarchical framework and to classify existing and new component naming terms within it.

5.3. BACKGROUND INFORMATION

The motivation for developing a component ontology for systems design is analogous to that of the museum curator who archives artifacts from the universe around us as a repository of knowledge about those artifacts. Research in the field of artificial intelligence (AI) known as knowledge capture and representation is closely related to the work reported here. In general, an ontology is a philosophical theory about the nature of existence, but AI researchers have adapted the term to describe “a shared and common understanding of some domain that can be communicated between people and application systems” (Gruber, 1994). Neches, *et al.,* (1991) claim: “An ontology defines the basic terms and relations comprising the vocabulary of a topic area.”

5.3.1. Artifact Classification. In this paper the view of an ontology is taken as a construct for the classification of knowledge:

> “An ontology may take a variety of forms, but necessarily it will include a vocabulary of terms, and some specification of their meaning. This includes definitions and an indication of how concepts are inter-related which collectively impose a structure on the domain and constrain the possible interpretations of terms.”(Uschold, 1998)

A rich source for information about artifact classification is found in the ontologies used by museums. Because museums are in the business of collecting, cataloging, and classifying the artifacts of human endeavor, their curators have spent considerable energy in devising systematic means of cataloging their collections. One of the tools employed in this classification is a lexicon. The most commonly used lexicon is the one developed by Chenhall (1978), who stated:

> “The lexicon…is based on the assumption that every man-made object was originally created to fulfill some function or purpose and, further, that original function is the only common denominator that is present in all of the artifacts of man, however simple or complex.”

In Chenhall’s view, the known (or assumed) function of an object represents the highest level of organizing principle upon which human-made artifacts can be classified and named. A logical system for naming objects consists of a ontology, or hierarchical ordering, based on three levels of relationships: 1) A controlled list of major categories, 2) A controlled vocabulary of classification terms, and 3) An open vocabulary of object names. Each of these levels is based on the function of the object:

- Major categories are a very limited set of easily remembered functional classes.
- Classification terms are carefully defined subdivisions of the major categories.
- Object names are the words used to identify individual artifacts.

The AI community takes a similar approach to component classification by using the function and form of a component as fundamental elements in its classification. The inclusion of function is a consistent theme in both the practical approach of Chenhall and the virtual approach of the AI community. The presence of component function in component naming is an important linkage between the theory of knowledge capture and representation and the theory of design. An understanding of function is integral to the design process (Pahl and Beitz, 1996; Otto and Wood, 2001); hence, a natural relationship between components and function must exist.

Another approach to classification comes from the Linnaean system of classifying species used in biology (Linnaei, 1937). Carolus Linnaeus began the classification of living species during the early 1700s. Originally organizing plants by their reproductive structures, Linnaeus laid the foundations for the modern organism classification, which later led to striking observations and evolutionary theories about the similarities between functional forms found between species in the natural world. In the Linnaean system, the two classes are the genus class and the species name; these are equivalent to the classification and object name within the Chenhall system. In Chenhall’s lexicon, the classifications are defined very clearly, while the object names are left open ended. This approach allows those interested in the lexicon to add to the collected knowledge contained therein. When used properly, a classification and an object name from Chenhall’s lexicon results in a name that is unique in all of humankind’s creations.

One difficulty in developing an ontology for components is classification consistency. For example, does a long slender two-force member describe a link, a beam, or a shaft? Stahovich, *et al.*, (1993) claim that the fundamental ontology for mechanical devices should be based on object behavior not structure. Paredis, *et al.*, (2001) suggest that a complete description of a component requires the addition of form to the classification, where form specifies a particular instantiation of a component, e.g., a part number for a motor. Both approaches imply that behavior is a key element in classifying mechanical components. Does this clear up the issue of the long slender two force member? The behavior of this component is describable using the mathematical representation of the states of a device (Pahl and Beitz, 1996). Modeling using the state representation of the component leads to an input/output relationship. Input/output relationships at a more abstract level are, by definition, the function of a component, device, or system. “A function of a product is a statement of a clear, reproducible relationship between the available input and the desired output of a product, independent of any particular form (Otto and Wood, 2001).” In the case of the long slender two force member, the input/output relationship is to transmit force, where transmit force is a function taken from the Functional Basis of Hirtz, *et al.*, (2002). Hence, it is proposed that the function of a component is the fundamental ontology for components.

5.3.2. Observations. In this work, common ground is found between the goal of a basis set of component names in systems design and Chenhall’s lexicon for classifying human-made artifacts. Because most components used in systems design are indeed human-made artifacts, they should be describable in the lexicon of Chenhall. Unfortunately, the lexicon does not include all possible artifact names, in fact “Artifacts originally created to be a physical part of some other object have, in most cases, been excluded from the lexicon” (1978). In terms of design, “artifacts originally created to be a physical part of some other object…” describe components.

Similarly, electro-mechanical devices share characteristics with living organisms that make the creation of a classification system analogous to the Linnaean classification, like having distinct observable form and function traits, varied levels of complexity, and a potential for partial overlap with traits from distinctly different components.

Since components cannot be adequately described in either Chenhall's lexicon or the Linnean classification, this function-based component ontology for systems design is proposed in order to establish a vocabulary of terms and a set of specifications for their inter-relationship. Therefore, similar to the way the Linnaean classification system has spawned an international code to ensure uniqueness and distinctness in naming biological terms, it is anticipated that the naming of new component terms under a component ontology should employ similar procedural guidelines.

5.4. THE CLASSIFICATION HIERARCHY

Although not completely analogous, systems and their components share many traits with animals that make classification challenging. Originally, animal classifications were primarily based on visual observations of morphological similarity. More recently, biologists have used molecular and biochemical data in addition to morphological data to identify evolutionary links and classify animals under what is thought to be a more accurate binary tree structure known as cladistics (Hennig, 1979). Components are not evolutionary in the same sense that animals evolve from what is commonly thought to be a series of branching points, and the goal of classification in this research is focused more on the practical use of the proposed hierarchical ontology. For this reason, we have chosen to initially begin with a function-based framework for the component classification hierarchy. The hierarchical framework was initially established from the notion that device function is an integral and critical characteristic of a component from the perspective of concept selection during the design process (Pahl and Beitz, 1996; Otto and Wood, 2001). As a starting point, the list of primary and secondary level function terms from the Functional Basis (Hirtz, 2002), discussed in Section 2.2., were used to designate the primary and secondary levels of the component framework.

5.4.1. Establishing the Hierarchy. In order to begin placing existing terms (Kurtoglu, *et al.*, 2005) into the framework, the functional traits of each device term needed to be established, where a device (component) is defined as having "input and output ports through which it is connected to another device (component)" (Kitamura and Mizoguchi, 2003). The functional traits of each component term were determined by analyzing the individual components housed within the repository of product information and categorized under that component term. The black box functionality for each component term was defined by identifying the most commonly occurring subfunction (function-flow combination) assigned to each of the components classified under that term in the repository.

5.4.2. Placing Existing Component Terms into the Hierarchy. Function templates for each component term (see Figure 5.1) were generated to show the functions assigned to components within a given classification. In nearly every case, a component term would have a single function that was common among all components classified under that term. Exceptions included components that had errors resulting from entering the data into the repository (e.g. no conceptual functions were assigned to an electric motor) and components that are classified as *Provisioners* where the functions *Store* and *Supply* were nearly always both included as conceptual functions. The functional information was then used to locate the appropriate placement for the component term within the hierarchical framework.

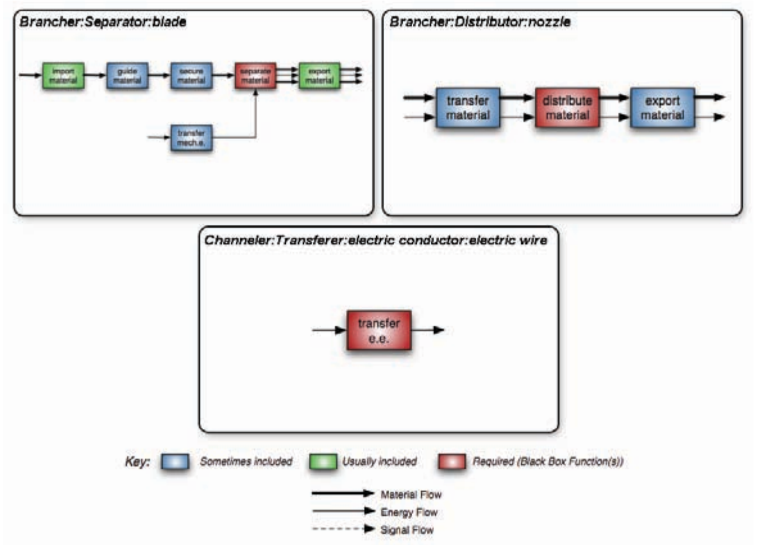


Figure 5.1. (Above) Function templates were used to help establish the functional characteristics of each component term. The templates were constructed using function and flow information entered into the web-based repository described in Section 2.

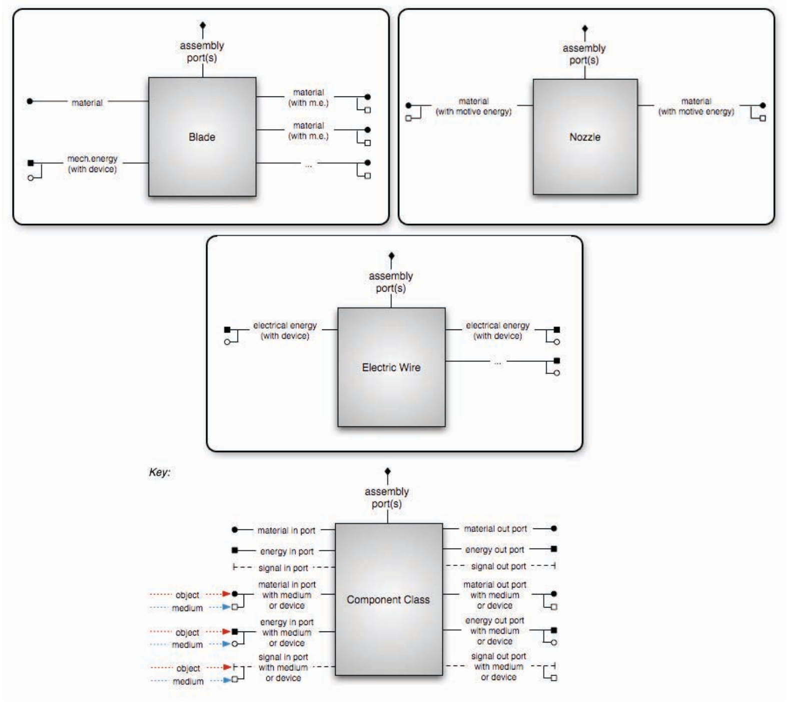

assembly port(s)
material
material (with m.e.)
material (with m.e.)
Blade
mech.energy (with device)
...
assembly port(s)
material (with motive energy)
Nozzle
material (with motive energy)
assembly port(s)
electrical energy (with device)
Electric Wire
electrical energy (with device)
...
Key:
assembly port(s)
material in port
energy in port
signal in port
material in port with medium or device
energy in port with medium or device
signal in port with medium or device
object
medium
Component Class
material out port
energy out port
signal out port
material out port with medium or device
energy out port with medium or device
signal out port with medium or device

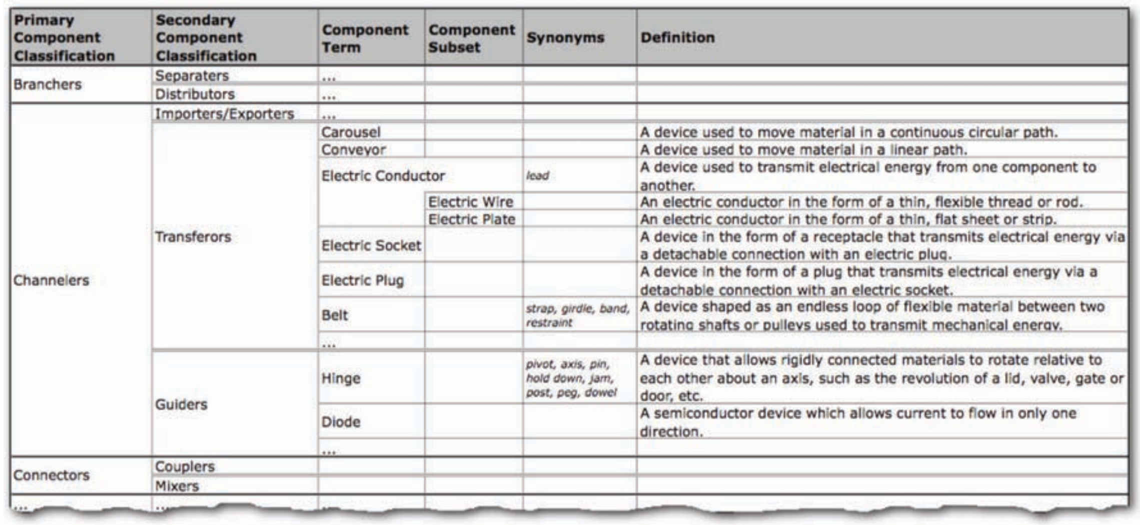
Figure 5.2. Port templates used to help establish the functional characteristics of each component term and to help create distinct definitions for each. Ports are indicated by lines into and out of the component box. Circles represent material flow ports, squares represent energy flow ports, and dashed lines with a vertical terminus represent signal flows. Components classes with members exhibiting variable numbers of repeating object ports are indicated by an output flow with ellipses (...), as shown for the electric wire.

In addition to function templates, templates that describe the major flows through a component were also established for each component term (Figure 5.2). The set of function and port templates for each of the components classified at this time can be found in Appendix B. In creating the port templates, the following port definitions were utilized:

> ***Object port:*** A device port through which a flow (material, energy, or signal) enters and then travels through the device from the input port to the output port and is processed by the device (Hirtz, 2002; Kitamura and Mizoguchi, 2003).
>
> ***Medium port:*** A device port through which a flow (material, energy, or signal) enters and then travels through the device from the input port to the output port while holding an object and enabling it to flow through the device (e.g. water can act as a medium carrying hydraulic energy as an object through a device) (Hirtz, 2002; Kitamura and Mizoguchi, 2003).
>
> ***Assembly port:*** A device port that acts only as a mating surface to support the weight or stabilize the position of the device.

Flow information contained in the repository was used to identify all ports of a particular component. This information was then generalized to create a standard template for the component term group. For this research, port templates only include the object and medium flows that are directly relevant to the function the component performs (e.g. the *material* separated by a *blade* and the *mechanical energy* used during the separation); waste flows, undesired flows, and reaction flows were not included (e.g. any *thermal* or *acoustic energy* that may result from a *blade* interacting with a *material* it is separating). Additionally, since they are not used at this point to help classify a component term, assembly connections were generalized into a single assembly port in each component template. Component term definitions within the hierarchical ontology were standardized using flow information from the port templates in addition to common morphological characteristics of the components within a single group. The previously developed list of component terms was refined to adhere to the newly developed rigorous classification structure (see Table 5.1 for an excerpt of the full list found in Appendix C).

Table 5.1. An excerpt of component terms and definitions organized using the proposed hierarchical ontology (the full component list may be found in Appendix C).



Primary Component Classification	Secondary Component Classification	Component Term	Component Subset	Synonyms	Definition
Branchers	Separaters	...			
	Distributors	...			
Channelers	Importers/Exporters	...			
	Transferors	Carousel			A device used to move material in a continuous circular path.
		Conveyor			A device used to move material in a linear path.
		Electric Conductor		*lead*	A device used to transmit electrical energy from one component to another.
			Electric Wire		An electric conductor in the form of a thin, flexible thread or rod.
			Electric Plate		An electric conductor in the form of a thin, flat sheet or strip.
		Electric Socket			A device in the form of a receptacle that transmits electrical energy via a detachable connection with an electric plug.
		Electric Plug			A device in the form of a plug that transmits electrical energy via a detachable connection with an electric socket.
		Belt		*strap, girdle, band, restraint*	A device shaped as an endless loop of flexible material between two rotating shafts or pulleys used to transmit mechanical energy.
		...			
	Guiders	Hinge		*pivot, axis, pin, hold down, jam, post, peg, dowel*	A device that allows rigidly connected materials to rotate relative to each other about an axis, such as the revolution of a lid, valve, gate or door, etc.
		Diode			A semiconductor device which allows current to flow in only one direction.
		...			
Connectors	Couplers				
	Mixers				
...	...				

The individual component terms and associated definitions represent the different “species” of components. Definition of these terms is critical to the usefulness of the ontology proposed. In animal classifications, disagreements exist over how narrowly to define different species, i.e. whether to identify species based primarily on minor differences (splitters (Merriam-Webster, 2005)) or major differences (lumpers (Merriam-Webster, 2005)). Similar questions become valid when defining new or existing component terms. For example, should an axle and a drive shaft be classified under the same component term? Should a flexible hose be classified under a different component term than a rigid tube? In the case of the axle and drive shaft, these two components solve different functionality and would, therefore, be placed under different branches of the proposed ontology. The flexible hose and rigid tube are functionally similar, so a decision must be made about whether to group them together under a broad definition or separate them into more specific groups. When defining terms, effort was made to determine whether a new (separate) definition would be beneficial from the perspective of a designer in the early conceptual stages of design, e.g. deciding whether to use a flexible vs. a rigid tube to *transfer* a *material* would be less useful when initially generating concepts than deciding whether to use a *tube* vs. a *conveyor*. To help evaluate whether terms were defined at a low enough level of detail, additional consideration was made as to whether generalities of performance could be made across a component term to help evaluate ideas early in the conceptual phase of the design process.

In general, the initially selected function-based framework worked well to help classify the existing component terms, with two notable exceptions. First, as briefly mentioned before, in nearly all cases of a component solving the function of *store*, the function of *supply* was also included. For this reason, the secondary level of the component hierarchy was refined to eliminate the separate designations of a *Storer* and a *Supplier* and instead include the secondary designation of a *Material* or *Energy Supplier*. Secondly, under the primary level term *Convert* in the Functional Basis exists a single secondary level term *Convert*. To eliminate redundancy in the proposed hierarchical ontology, the secondary level term *Converters* was replaced with designations of a

Material, *Energy*, or *Signal Converter*. The complete component hierarchy can be found in Figure 5.3.

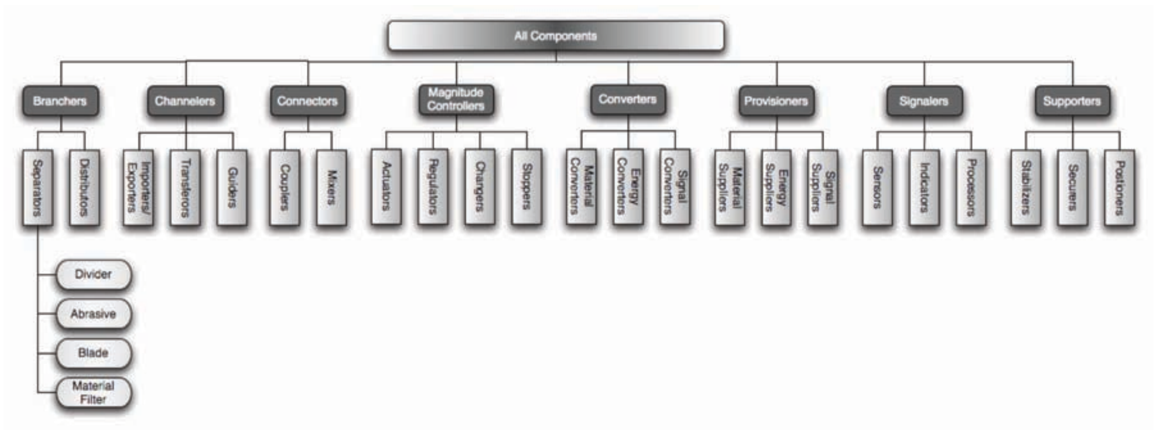


Figure 5.3. The proposed function-based hierarchical ontology structure. Only the component terms for the class of Separators are shown.

5.5. CLASSIFYING COMPONENTS USING THE ONTOLOGY

A rigorous procedure was established in order to determine under which component term a previously unclassified component should be grouped within the established hierarchical framework. The procedure developed is as follows:

1. Define the system boundary of the device.
2. Identify all input and output ports of the device across the system boundary defined in Step 1.
3. Classify each port as an
    a. *Object port:* A device port through which a flow (material, energy, or signal) enters and then travels through the device from the input port to the

output port and is processed by the device (Hirtz, 2002; Kitamura and Mizoguchi, 2003).

   b. *Medium port:* A device port through which a flow (material, energy, or signal) enters and then travels through the device from the input port to the output port while holding an object and enabling it to flow through the device (e.g. water can act as a medium carrying hydraulic energy as an object through a device) (Hirtz, 2002; Kitamura and Mizoguchi, 2003).

   c. *Assembly port:* A device port that acts only as a mating surface to support the weight or stabilize the position of the device.

4. Identify the black box functionality of the device and the object flow(s) that it acts on. When defining the black box functionality, the functional purpose of the device should be identified versus the functional embodiment of the device (i.e. the function selected should answer the question "what does this device do?" instead of the question "how does this device work?") For instance, the functional purpose of a friction brake is to "stop rotational energy" and it does this by "converting rotational energy to thermal energy". In this case, the black box functionality of the brake would be to "stop rotational energy."

5. Locate device placement in classification hierarchy.

   a. Label device using appropriate term.

   b. If no existing term is suitable, create a new term under the relevant hierarchical category. Generate a definition precisely defining the form of the device in a manner that clearly distinguishes the new device from the other components located under the same functional class.

5.6 ANALYTICALLY DERIVED DESIGN STRUCTURE MATRIX (DSM)

In the concept generation algorithm presented in Section 3, the first pass “filter” for pruning the space of possible solutions utilizes component–component compatibility information in the form of Design Structure Matrix (DSM). Although there are many considerations to take into account when determining full compatibility between components (e.g. spatial characteristics, energy domain constraints, etc.), a DSM contains general “go” or “no go” component compatibility information from products that have previously been designed and, in most cases, commercially manufactured. Therefore, although specific compatibility parameters are not enumerated, general compatibility between components can be utilized to implicitly weed out solutions that contain component connections that have not been embodied before—whether due to incompatibility issues or other design rationale.

To extend the information generated purely from repository data, a DSM was analytically constructed using the component templates described in Section 5.4.2. For each of the component templates shown in Appendix B, in and out energy, material, and signal ports were analyzed. Compatibility with another component was identified as possible if any of the ports could be “connected”, i.e. if the flow types were the same. For instance, Figure 5.4 shows the component port template for an airfoil.

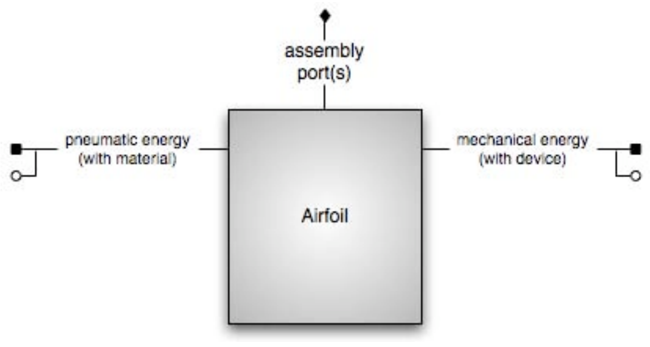

assembly port(s)
pneumatic energy (with material)
mechanical energy (with device)
Airfoil

Figure 5.4. Component port template for an airfoil.

At this time, no component is contained in the repository that can be classified as an airfoil. However, using the port information contained in the template, an airfoil accepts pneumatic energy as an input and outputs mechanical energy. Looking through the templates of the other currently classified terms, port flow comparisons can be made with each of them. Thus, if a component outputs pneumatic energy, it is deemed possible for that component to be compatible with an airfoil, and a “1” is entered into the corresponding DSM cell. Similarly, if a component accepts mechanical energy as an input, it would also be deemed as a potential compatible component and have a “1” placed into the corresponding DSM cell. Zeros, “0”s, are placed in the DSM if a component has no potentially compatible ports with the airfoil. Sorting through the list of components using this procedure identifies 41 different components with the potential to comprise a complete solution in conjunction with an airfoil. Varying on the functionality each design fulfills, several potential airfoil-compatible examples are shown in Figure 5.5.

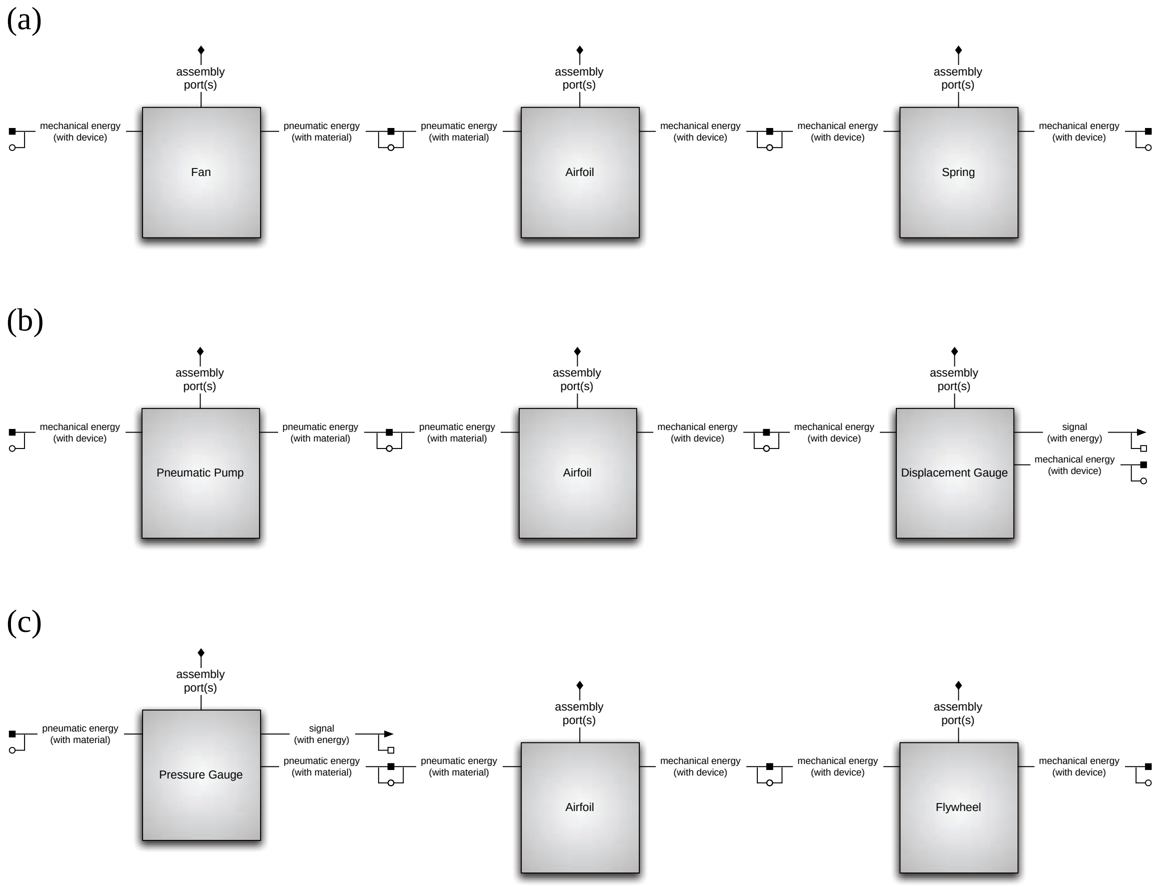

(a)
assembly port(s)
mechanical energy (with device)
Fan
pneumatic energy (with material)
pneumatic energy (with material)
Airfoil
mechanical energy (with device)
mechanical energy (with device)
Spring
mechanical energy (with device)
(b)
assembly port(s)
mechanical energy (with device)
Pneumatic Pump
pneumatic energy (with material)
pneumatic energy (with material)
Airfoil
mechanical energy (with device)
mechanical energy (with device)
Displacement Gauge
signal (with energy)
mechanical energy (with device)
(c)
assembly port(s)
pneumatic energy (with material)
Pressure Gauge
signal (with energy)
pneumatic energy (with material)
pneumatic energy (with material)
Airfoil
mechanical energy (with device)
mechanical energy (with device)
Flywheel
mechanical energy (with device)

Figure 5.5. Potential compatible components for an airfoil.

The analytical DSM, presented in Appendix D, identifies the set of compatible components for the current set of component terms. This compatibility has been determined solely using port information and without consideration for design rationale with the notion of enhancing the potential for innovative solutions to be derived. By extending the compatibility information beyond only those connections that have occurred before, the hope is that a new combination of components could be considered for a design and design rationale could then be reintroduced to investigate the feasibility of the new idea. In this way, truly original designs would be less likely to be banished by existing design biases regarding component compatibility.

6. EXPERIMENTS AND CASE STUDIES

6.1. INTRODUCTION

The research described in this section presents several verification experiments and case studies executed to test the ability and effectiveness of the proposed computational technology to automatically generate relevant conceptual solutions. First, several experiments are reported which test the validity of results returned when utilized by students in a structured design process for several design scenarios. Then, several case studies are shown illustrating the effectiveness of the concept generator in multiple design situations.

6.2 EXPERIMENT: UNDERGRADUATE INVESTIGATION, PART I

To qualitatively evaluate the practicality of using the concept generator to produce conceptual design variants early in the design process, four undergraduate researchers from the University of Texas at Austin and the University of Missouri-Rolla were directed to complete several different activities. In the first activity, the students were instructed to qualitatively compare manually generated concepts against automatically generated design solutions produced using the list-based concept generator, described in Section 4.2, for three original design scenarios. The data collected by the students during this methodological comparison were later studied quantitatively, and the results can be found in Section 6.2.2. Further activities investigated the robustness of solutions returned by the concept generator against variations in the functional modeling chains used to seed the generation of concepts, including permutations and omissions of subfunctions. The following sections describe, in detail, both the qualitative and quantitative comparisons of the student-derived design solutions to the solutions automatically generated by the concept generator as well as the robustness studies that the undergraduate researchers engaged in during their activities.

6.2.1. Experimental Setup. The following subsections describe the experimental procedure the students followed during the course of this experimental study.

6.2.1.1. Methodological Comparison. To evaluate the validity of the design solutions returned by the concept generator, the undergraduate researchers first investigated how the automatically produced concept variants compared to concepts that they had generated manually using a morphological matrix approach. In order to do this, the students looked at three different design scenarios that investigated concepts produced for an original design. The students were instructed to complete the manual concept generation activities for each design scenario prior to exploring any results generated by the concept generator software to avoid any unintentional biasing of results.

The flowchart in Figure 6.1 shows an overview of the structure of activities.

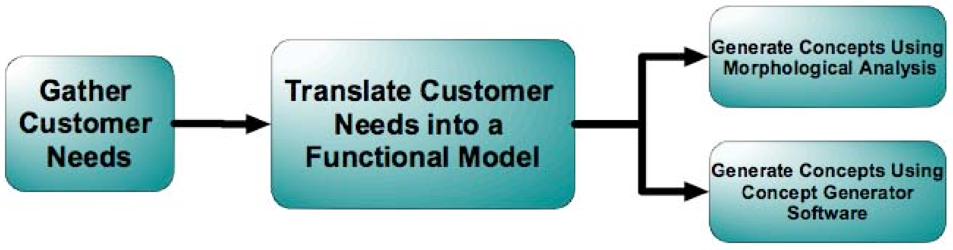

Gather Customer Needs
Translate Customer Needs into a Functional Model
Generate Concepts Using Morphological Analysis
Generate Concepts Using Concept Generator Software

Figure 6.1. Flowchart of the activity structure for the concept generation methodological comparison.

For the methodological comparison, students generated design solutions for each of the original design problems described below.

- ***Hot or cold thermal mug:*** This original design entailed creating a thermal mug to be used either to keep a hot beverage hot or a cold beverage cold. The idea was to create a thermal mug that is superior to ones currently on the market that rely solely on insulating techniques to achieve thermal isolation. In other words, concepts needed to be generated that not only attempted to inhibit the transfer of heat, but also had the ability to add or remove heat to the beverage.
- ***Human powered power supply:*** For this original design, the students were instructed to design a human-powered power supply that could reasonably supply enough electricity consistently to power an audio-visual device or that could be used to recharge batteries.
- ***Wall climbing toy:*** In this original design scenario, a company has begun marketing a wall coating that contains ferrous micro-metal chips. This coating is “attractive” to magnetic devices and walls coated with this product “look” metallic. One potential marketing ploy for the company to increase sales of its coating product is to sell a toy that would operate on the vertical space of the walls (or even the ceiling). Thus, the undergraduate researchers were instructed to generate concepts for toy products that utilize walls covered with the coating as their play space. Since there are numerous types of potential toys for this new application, this call for products is fairly open ended. Broad requirements for the students to exhibit in their design included the ability to direct the toy

accurately to specific points on the wall, remain stationary while on the wall, be marketable to a broad customer segment, be lightweight, have a long-lasting power source, and be inexpensive and easy to set up.

Using the design steps shown in Figure 6.2, the undergraduate researchers produced functional models from the customer needs they established (from customer interviews) for each product.

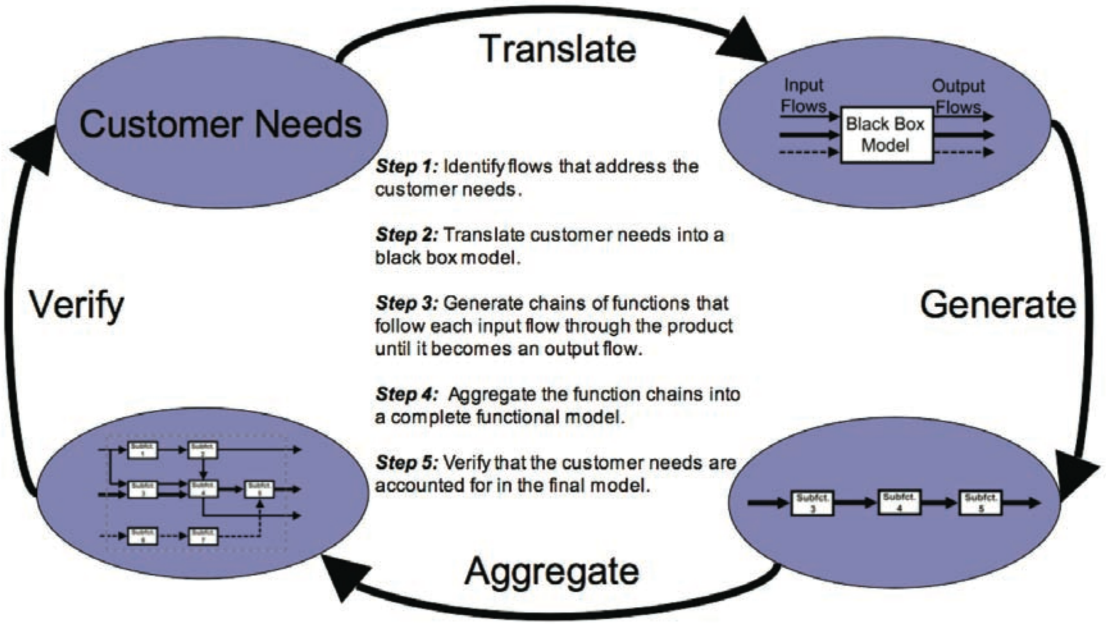

Translate
Customer Needs
Input Flows
Output Flows
Black Box Model
Step 1: Identify flows that address the customer needs.
Step 2: Translate customer needs into a black box model.
Step 3: Generate chains of functions that follow each input flow through the product until it becomes an output flow.
Step 4: Aggregate the function chains into a complete functional model.
Step 5: Verify that the customer needs are accounted for in the final model.
Verify
Generate
Aggregate

Figure 6.2. The students used the steps illustrated above to generate functional models for each product design scenario from the customer needs they established through customer interviews.

For the original design scenarios, the undergraduate researchers began by producing functional models for each product from customer needs (established from customer interviews) using the design steps shown in Figure 6.2. Once a functional model was generated, the students generated partial solutions for each product using a morphological matrix. Finally, the students assembled several complete solutions for each design from the corresponding morphological matrix, and produced design variant sketches as well as lists of components comprising each of their designs. To avoid pollinating the manually generated morphological matrices with ideas from the design repository, the undergraduate researchers completed all concept generation tasks for the original and redesign scenarios before moving on to generate designs solutions using the concept generator software. The final step of the methodological comparison was to generate conceptual variants for each design using the concept generator software. Since the software user input was limited at this time, functional models had to first be separated into sequential (non-parallel) chains, with instructions given to the undergraduate researchers to experiment with how they chose to dissect the functional models for entry into the program. The students were instructed to compare the results generated by the software with those they had generated manually and make notes of any thoughts they had on the results produced for the chains they had entered. All design solution chains generated via the software were saved to text files that included the input function chain that was used to generate that set of concept variants.

As an extension to the methodological comparison study performed by the undergraduate researchers, the original design solutions generated by the students were later compared quantitatively to those generated by the concept generator from the design repository knowledge. Since the complete set of student design solutions was contained in a morphological matrix while the complete set of solutions produced by the concept generator consisted of lists of compatible solutions, making direct comparisons of the solutions was difficult to achieve. In order to make more quantitative comparisons, the design solutions generated by the students were translated into lists of compatible solution chains that could more easily be compared to those generated by the concept generator from the repository data, see Figure 6.3. Additionally, the results returned by the concept generator were separated out into morphological matrices that could more directly be compared to the morphological matrices manually generated by the students, also shown in Figure 6.3.

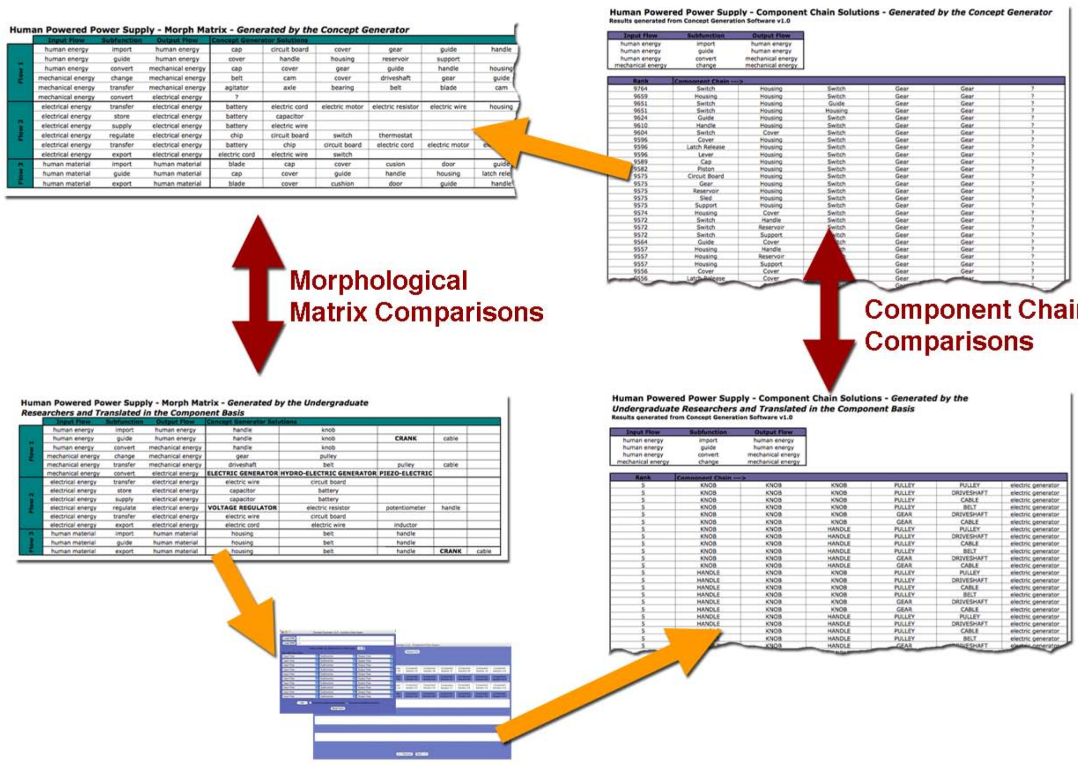

Human Powered Power Supply - Morph Matrix - Generated by the Concept Generator
Human Powered Power Supply - Component Chain Solutions - Generated by the Concept Generator
Morphological Matrix Comparisons
Component Chain Comparisons
Human Powered Power Supply - Morph Matrix - Generated by the Undergraduate Researchers and Translated in the Component Basis
Human Powered Power Supply - Component Chain Solutions - Generated by the Undergraduate Researchers and Translated in the Component Basis

Figure 6.3. The lists of component chains returned by the concept generator were transformed into morphological matrices that could more easily be compared to the morphological matrices produced by the undergraduate researchers. Similarly, the morphological charts produced by the students were transformed into lists of feasible component chains.

The lists of student generated compatible solution chains were created by first manually translating each morphological matrix generated by the students into a function-component matrix (FCM) for each product. Next, a design structure matrix (DSM) was generated by inspection for each product. In other words, the DSM cell entries defining solution compatibility were manually entered for each design solution combination, e.g. a “battery” can be connected to a “wire” so a “1” would be placed in the corresponding DSM cell to indicate compatibility. Conversely, a “bubble” is unlikely to be connected to a “levee” so a “0” would be placed in the corresponding DSM cell to indicate incompatibility. The manually constructed FCM and DSM for each product were then used in the concept generator to seed the solutions produced for an entered function chain. This, in effect, produced a list of design solution chains with incompatible solutions filtered out. The concept generator derived morphological matrices were produced by dissociating each component solution from the chain of compatible components and recording the unique solutions produced for each function entered.

Finally, the student-derived morphological matrices were classified using the component naming basis where applicable, in order to help facilitate comparisons with the concept generator design solutions. This translation also helped identify and combine similar design solutions generated by the students, e.g. under the component naming scheme a “soda container” a “coffee pot” and a “water tank” would be classified as different instantiations of a “reservoir”. Grouping the student solutions under the component naming basis had the effect of grouping similar solutions and identifying ideas generated by the students that either need a classification under the Component Basis (e.g. electric generator) or were outside the black box boundary of the design scenario (e.g. fountain machine). After the terms were translated into the Component Basis, new morphological matrices, FCMs, DSMs, and sets of compatible solution chains were generated for comparison.

6.2.1.2. Returned Results Robustness Investigation. The undergraduate researchers next investigated the effect of how various permutations in the user-input function chain impacted the conceptual component chains returned by the concept generator software. Figure 6.4 gives an illustration of how a sample chain of functions might be permuted for investigation in this activity. To complete this task, the students extracted function chains from functional models they had generated for products dissected during an earlier activity. The undergraduate researchers next determined permutations in function adjacency that would still satisfy the functional requirements of the product and entered each permutation into the concept generator software. Again, the students were instructed to make notes of any thoughts they had on the results produced for the chains they had entered. All design solution chains generated via the software were saved to text files that included the input function chain that was used to generate that set of concept variants.

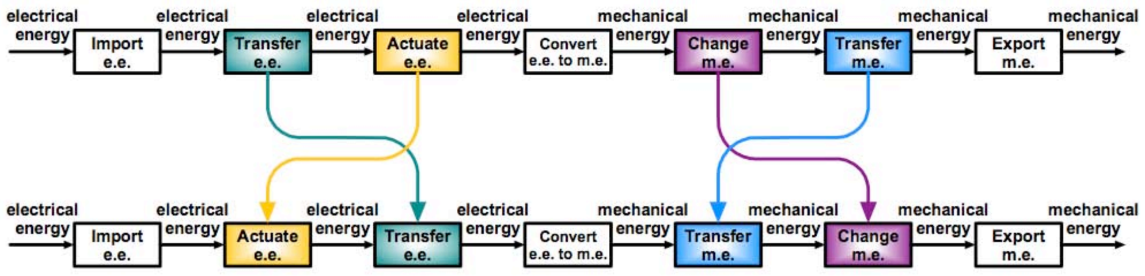


Figure 6.4. This activity investigated the effect of chain permutation on the conceptual results returns by the concept generator software.

6.2.1.3. Functional Model Variation Effects. The final activity focused on investigating the effect that variations in functional modeling generation might have on the results returned by the concept generator software. This activity, along with the robustness investigation described in Section 6.2.1.2., seeks to explore how dissimilarities in functional models produced by different designers might impact the solutions produced by the concept generator software. In particular, in this activity, the students looked at how the insertion or deletion of "minor" or "assumed" functionality impacted the results generated. For instance, one person may produce the conceptual functional model shown in Figure 6.5a, whereas another person may include more specific functionality that deals with the "transition" from one critical function to the next, such as the specific transfer of energy, as shown in Figure 6.5b.

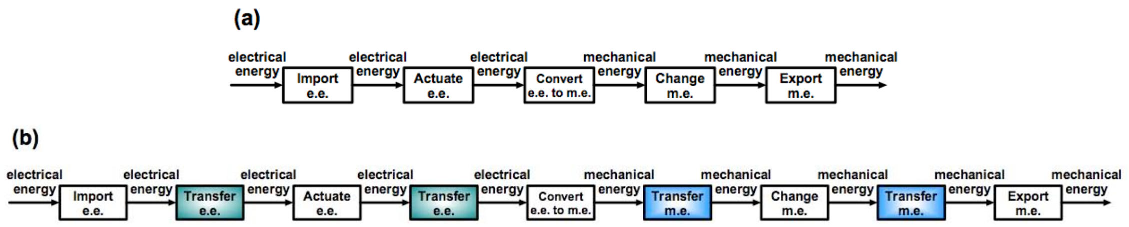


Figure 6.5. (a) A person may omit implied functions a product needs to exhibit while deriving a functional model. (b) A different person may be more explicit and include functional "transitions" in a functional model. This activity investigates the software results returned by function chains with slight variations in functionality.

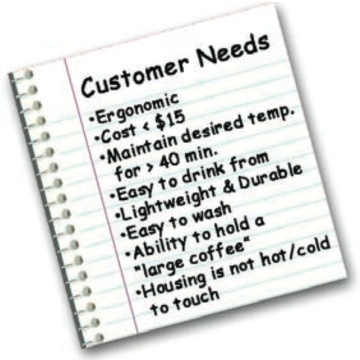
The undergraduate researchers were instructed to think about which functions might be considered to have "assumed" or "minor" functionality by a designer. Next, the students extracted function chains from the functional models generated for previously dissected products and for the original design activities presented in Section 6.2.1.1. that either already had or could include these "minor" functions. Finally, the undergraduate researchers compared the concepts produced by the software for chains with and without the “minor” functions included. As in the previously described activities, the students were instructed to make notes of any thoughts they had on the results produced for the chains they had entered, and all design solution chains generated via the software were saved to text files that included the input function chain used to generate that set of concept variants.

The next section presents a summary of the results produced by the undergraduates during the methodological comparison, with example results from the hot/cold thermal mug design included.

6.2.2. Results from the Experimental Study. The following sections present the results from the study of the methodological comparison as well as the results from the robustness and variation study activities described. Results from the undergraduate researchers’ evaluation activities indicated that manually generated concepts were completely encapsulated in the concept variant results returned by the software for the investigated design scenarios. In addition, with a few notable exceptions, the concept generator consistently averaged a larger quantity of feasible solutions for each subfunction than those produced manually by the students. Furthermore, results from the software-generated conceptual designs for function chains varied by permutation or omission indicated that similar concepts were returned for seed function chains with minor variations.

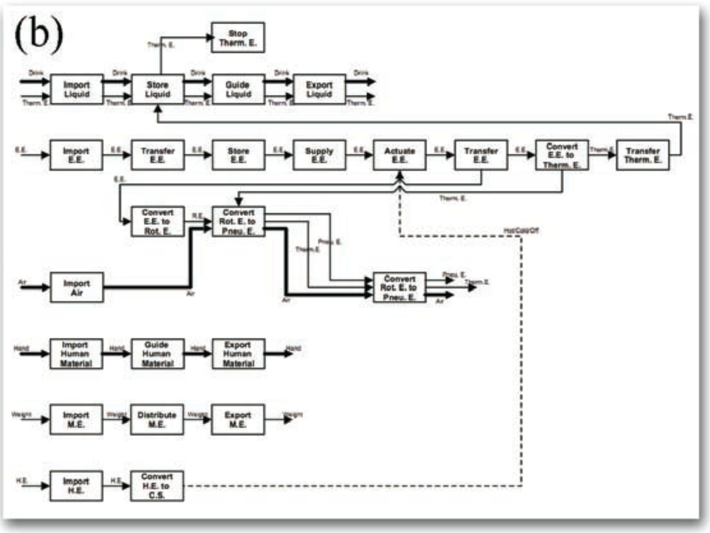
6.2.2.1. Methodological Comparison Results. For the methodological comparison, the undergraduate researchers manually developed original design solutions for the thermal mug, human-powered power supply, and wall-climbing toy design scenarios. They began by conducting interviews to collect customer need data for each original scenario. Next, the students used the customer needs to establish a functional model for each product using the method previously summarized in Figure 6.2. Using the subfunctions from the functional models, the undergraduate researchers manually constructed morphological charts to generate multiple partial solutions for each discrete functional element the design needed to embody using brainstorming techniques. Finally, the students selected a partial solution for each subfunction and sketched a complete concept capable of solving the given design problem. This last step was repeated several times to produce multiple concept variants for each design scenario. Figure 6.6 gives a summary of the data manually generated by the undergraduate researchers for the thermal mug design scenario described in Section 6.2.1.1.

(a)

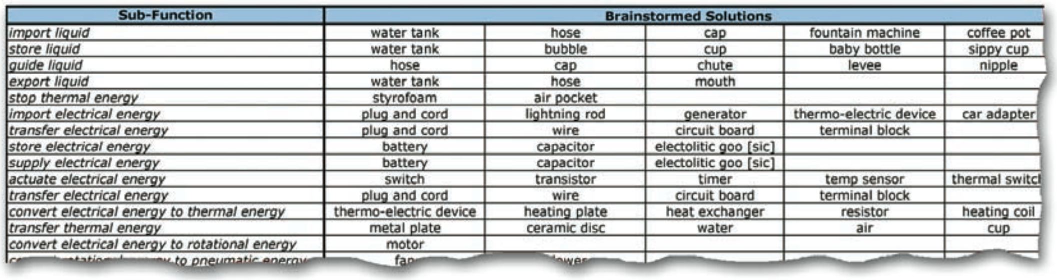


Customer Needs
- •Ergonomic
- •Cost < $15
- •Maintain desired temp. for > 40 min.
- •Easy to drink from
- •Lightweight & Durable
- •Easy to wash
- •Ability to hold a "large coffee"
- •Housing is not hot/cold to touch

(b)



(c)



Sub-Function	Brainstormed Solutions				
import liquid	water tank	hose	cap	fountain machine	coffee pot
store liquid	water tank	bubble	cup	baby bottle	sippy cup
guide liquid	hose	cap	chute	levee	nipple
export liquid	water tank	hose	mouth		
stop thermal energy	styrofoam	air pocket			
import electrical energy	plug and cord	lightning rod	generator	thermo-electric device	car adapter
transfer electrical energy	plug and cord	wire	circuit board	terminal block	
store electrical energy	battery	capacitor	electolitic goo [sic]		
supply electrical energy	battery	capacitor	electolitic goo [sic]		
actuate electrical energy	switch	transistor	timer	temp sensor	thermal switc
transfer electrical energy	plug and cord	wire	circuit board	terminal block	
convert electrical energy to thermal energy	thermo-electric device	heating plate	heat exchanger	resistor	heating coil
transfer thermal energy	metal plate	ceramic disc	water	air	cup
convert electrical energy to rotational energy	motor				
to pneumatic energy	fa	lowe			

(d)

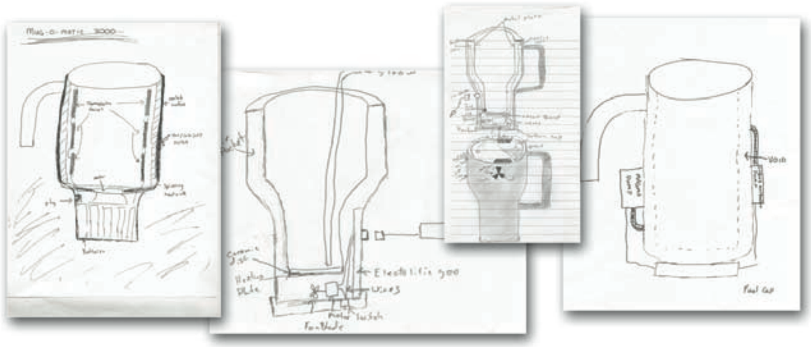


Figure 6.6. The students began the methodological comparison for the thermal mug by generating (a) customer needs, (b) functional models, (c) morphological charts, and (d) complete concept sketches.

After generating similar sets of data for each of the original and redesign scenarios, the undergraduate researchers divided the functional models they developed during the design process into single non-branching chains of functions and entered the chains separately into the concept generator software. In the case of the thermal mug design, the hypothetical functional model was broken into 8 function chains. Next, they compared the concepts returned by the concept generator against the complete concepts they had assembled from their morphological charts. The undergraduate researchers found that every flow chain they were able to gather results from returned at least one concept extremely similar to their manually developed concepts, with most of the matched solutions occurring toward the top of the ranked list of returned component chains. If we first classify the students' brainstormed solutions under the same Component Basis classification scheme that the concept generator uses to return components, the similar matches become identical, as shown in Figure 6.7. Each of the original and redesign scenarios resulted in successful comparisons that were similar to the thermal mug design example shown.

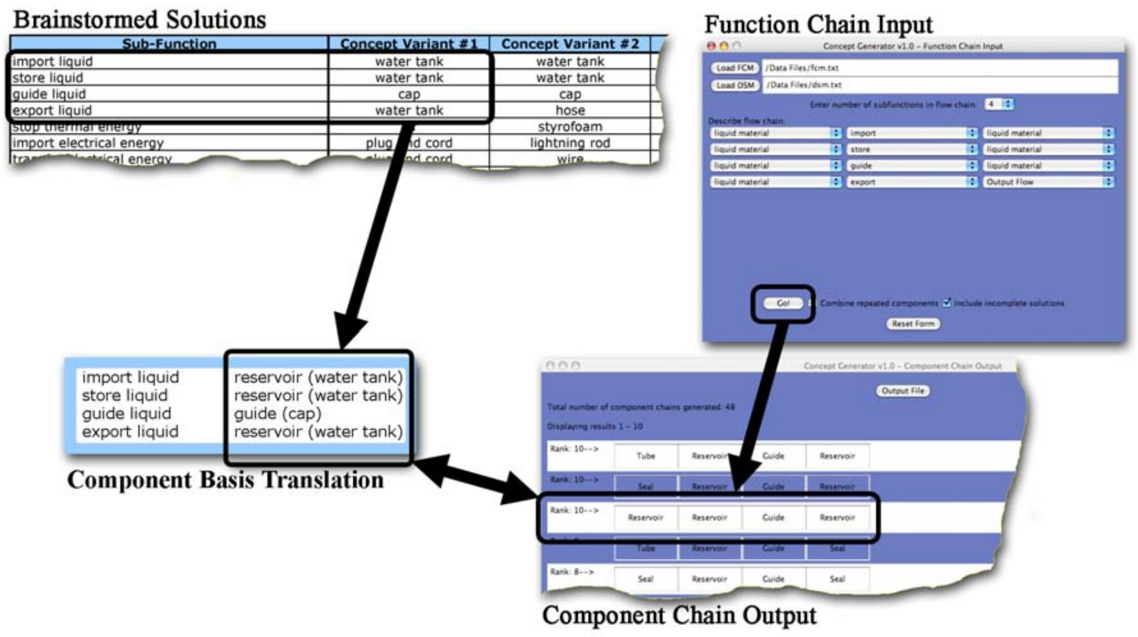


Figure 6.7. The students found nearly all of their manually generated concepts from their complete design solutions matched up with top-ranked solutions returned by the concept generator.

The lists of design solutions produced by the concept generator were saved as text files. Once the student generated design solutions had been combined into lists of feasible design solution chains and the software generated design solutions had been distilled into morphological matrices, numerous observations could be made regarding the quality and quantity of solutions produced by each method. Looking at the total number of distinct design solutions generated during the original design scenarios, on average, the concept generator produced more design solutions per subfunction than the students produced manually (6.85 vs. 2.45 as shown in Table 6.1). For all observations, a student generated partial design solution was considered unique if no other solution listed for the same subfunction was classified the same under the component naming scheme or if it did not fit any of the current component naming classifications. In other words, a design solution (e.g. an “electric wire”) would be considered unique even if was listed as a solution to multiple subfunctions in the morphological matrix, e.g. an “electric wire” may be listed as a solution to both the subfunction “import electrical energy” and “transfer electrical energy”. In this situation, the “electric wire” would be counted twice in a design solution count; once as a solution to “import electrical energy” and once as a solution to “transfer electrical energy”.

Table 6.1. Summary table showing the number of solutions generated for each original design scenario. The number of subfunctions included in each morphological matrix and the average number of solutions per subfunction for all design scenarios together is also shown.

	Solutions			
	Student Generated	**Student Generated (Using Component Basis Classifications)**	**Software Generated**	**Subfunctions**
Power Supply	51	43	150	15
Thermal Mug	93	79	152	28
Wall Climber	26	25	109	17
Totals	**170**	**147**	**411**	**60**
Average # of Solutions Per Subfunction	**2.83**	**2.45**	**6.85**	

Tables 6.2a-c give a more detailed breakdown of the number of solutions and feasible solution chains produced by each method for each specific original design scenario. Data within these tables are organized by the flow chains that were entered into the concept generator to produce corresponding chains of compatible partial solutions.

From these tables, we can see that the average number of solutions produced per subfunction for nearly every flow is higher for the concept generator group vs. the student generated group of solutions. Correspondingly, the total number of compatible solutions produced by the concept generator from the repository of design knowledge is typically greater than those produced by the students, with a few notable exceptions. First, in Table 6.2a, we can see that no complete solutions were assembled by the concept generator for “Flow 1” in the human-powered power supply design scenario. This observation stems from the fact that, at this time, no component in the design repository solves the subfunction “convert mechanical energy to electrical energy”. Similarly, in Table 6.2c, the lack of solutions for “Flow 2” in the wall climber toy design scenario results from the concept generator being unable to find a component solution to the subfunction “secure mechanical energy” that is historically compatible with the component found to solve the subfunctions “import mechanical energy” and “export mechanical energy”. Additionally, for “Flow 3” in the same scenario, no complete solutions were returned (although the student derived solutions were manually found contained in the design repository) because the subfunctions generated by the students were slightly varied from the models used when the components were entered into the repository database.

Table 6.2. Summary tables showing the number of subfunctions in each flow extracted from the full functional model, total number of solutions generated for all of the subfunctions in each flow, average number of solutions per subfunction within a flow, number of compatible solution chains able to be constructed (both partial and complete) to solve the flow, and the total number of solution chain combinations possible (both compatible and incompatible) for the (a) human-powered power supply, (b) hot or cold thermal mug, and (c) wall climbing toy design scenarios.

Power Supply Design Scenario		# Subfunctions	# of Solutions Generated	Avg. # Solutions/ Subfunction	Total Compatible Solutions Generated	Total Complete Compatible Solutions Generated	Total Possible Solution Combinations
Student Generated Morphological Matrix							
	Flow 1	6	20	3.33	270	270	360
	Flow 2	6	18	3.00	270	270	540
	Flow 3	3	13	4.33	75	75	75
Student Generated Morphological Matrix with Solutions Classified Using Component Name Classifications							
	Flow 1	6	17	2.83	96	96	128
	Flow 2	6	15	2.50	120	120	192
	Flow 3	3	11	3.67	45	45	45
Automatically Generated Solutions Using Concept Generator Software							
	Flow 1	6	76	12.67	46113	0*	330330
	Flow 2	6	37	6.17	921	916	7920
	Flow 3	3	37	12.33	790	790	1820

* At this time, no artifact in the repository solves the function "convert mechanical energy to electrical energy"

Thermal Mug Design Scenario		# Subfunctions	# of Solutions Generated	Avg. # Solutions/ Subfunction	Total Compatible Solutions Generated	Total Complete Compatible Solutions Generated	Total Possible Solution Combinations
Student Generated Morphological Matrix							
	Flow 1	4	21	5.25	86	86	630
	Flow 2	9	45	5.00	272161	266112	1209600
	Flow 3	1	2	2.00	2	2	2
	Flow 4	2	9	4.50	13	13	20
	Flow 5	3	9	3.00	27	27	27
	Flow 6	3	7	2.33	12	12	12
	Flow 7	3	9	3.00	6	6	7
	Flow 8	3	7	2.33	8	8	8
Student Generated Morphological Matrix with Solutions Classified Using Component Name Classifications							
	Flow 1	4	15	3.75	60	60	150
	Flow 2	9	37	4.11	32181	31280	201600
	Flow 3	1	2	2.00	2	2	2
	Flow 4	2	7	3.50	9	9	12
	Flow 5	3	9	3.00	27	27	27
	Flow 6	3	7	2.33	12	12	12
	Flow 7	3	7	2.33	3	3	5
	Flow 8	3	6	2.00	4	4	4
Automatically Generated Solutions Using Concept Generator Software							
	Flow 1	4	8	2.00	12	12	12
	Flow 2	9	44	4.89	7400	5761	131040
	Flow 3	1	2	2.00	1	1	2
	Flow 4	2	9	4.50	13	13	18
	Flow 5	3	38	12.67	790	790	1950
	Flow 6	3	26	8.67	208	208	468
	Flow 7	3	25	8.33	176	176	336
	Flow 8	3	15	5.00	25	24	60

Wall Climber Toy Design Scenario		# Subfunctions	# of Solutions Generated	Avg. # Solutions/ Subfunction	Total Compatible Solutions Generated	Total Complete Compatible Solutions Generated	Total Possible Solution Combinations
Student Generated Morphological Matrix							
	Flow 1	8	16	2.00	96	96	192
	Flow 2	3	3	1.00	1	1	1
	Flow 3	6	10	1.67	12	12	12
Student Generated Morphological Matrix with Solutions Classified Using Component Name Classifications							
	Flow 1	8	15	1.88	36	36	96
	Flow 2	3	3	1.00	1	1	1
	Flow 3	6	10	1.67	12	12	12
Automatically Generated Solutions Using Concept Generator Software							
	Flow 1	8	82	10.25	43979	42834	1782000
	Flow 2	3	0	0.00	0*	0*	0*
	Flow 3	6	27	4.50	294	0**	560

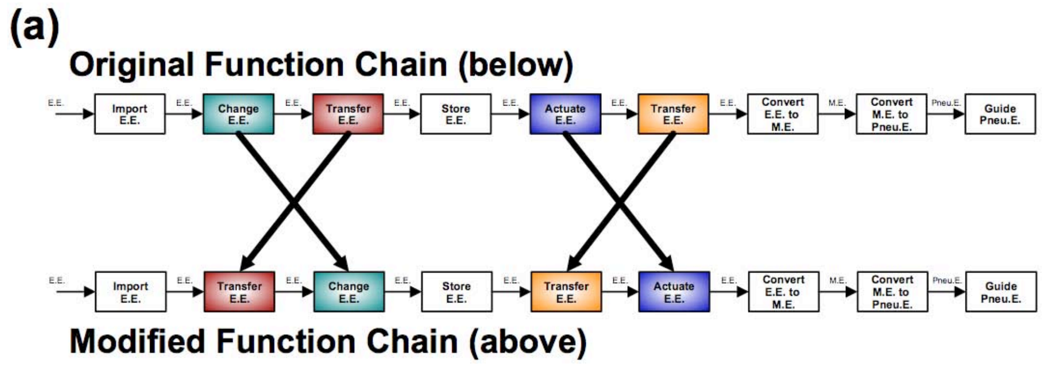
* At this time, no artifact in the repository solves the function "secure mechanical energy"
** Student generated functional model did not identically correspond with a suitable artifact in the repository generated FCM

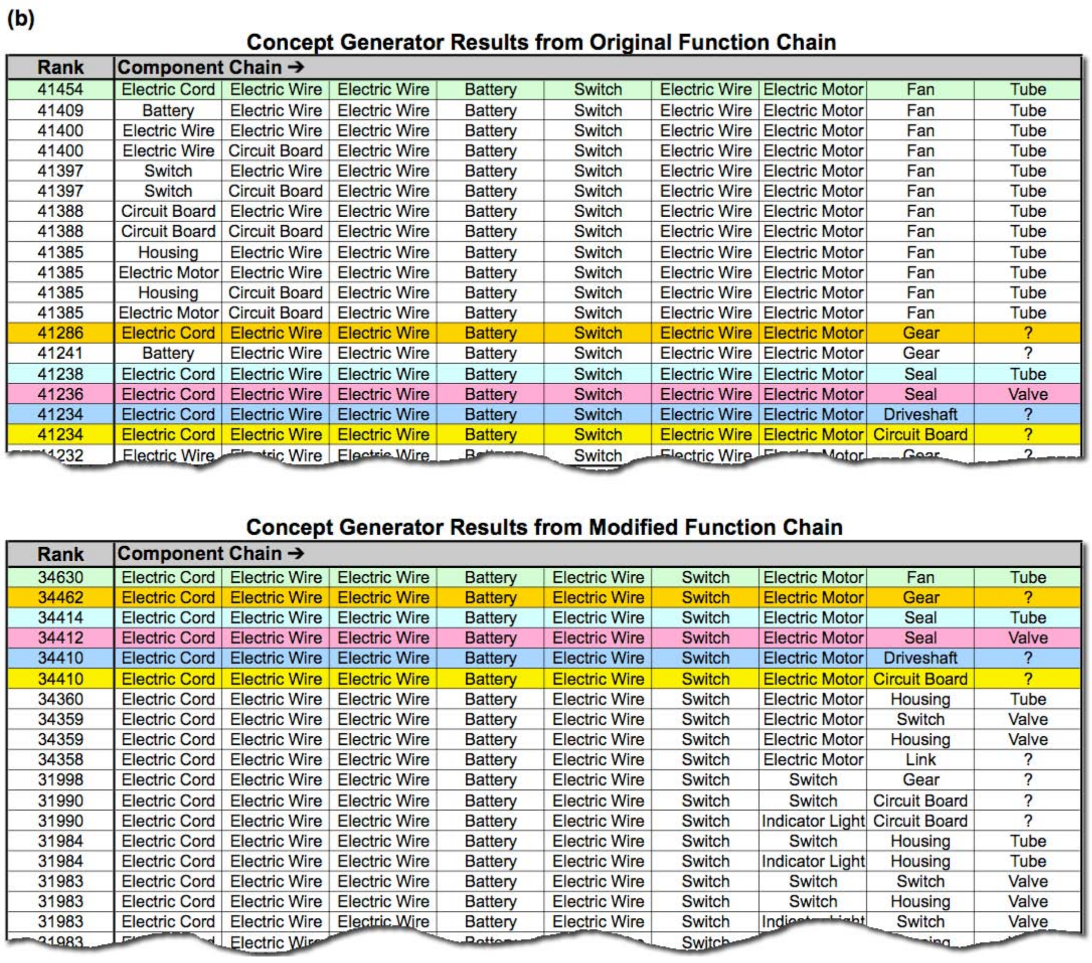
Since quantity of results is not the only concern when evaluating the usability of a design tool in concept generation, a comparison of the type of solutions produced by the concept generator against those produced by the students was also made. Table 6.3 shows a summary of the number of overlapping design solutions seen in both the student generated and concept generator derived morphological matrices. For instance, if we look at the human-powered power supply data, of the 43 distinct solutions produced by the students, 19 matched with solutions produced by the concept generator, meaning 44.19% of the student generated solutions were contained in the automatically generated solution set. Of the 24 remaining solutions produced by the students, 6 were not definable under the current version of the component naming scheme, including 3 solutions for the subfunction “convert mechanical energy to electrical energy,” for which no solutions currently exist in the design repository. Other times, the student generated solutions that did not match with solutions from the concept generator and were not classifiable under the component naming basis were either technically infeasible for the given design scenario, e.g. using a “bubble” to “store liquid material” or using a “levee” to “guide liquid material” for the thermal mug design, or too broad of an idea to be encapsulated by a single component in the design repository, e.g. using a “fountain machine” to “import liquid material”. Inspection of the results returned by the concept generator that did not overlap with the results generated by the students showed an overwhelming majority of viable alternatives. Only a few instances of obvious incorrect matches were identified, and each were linked back to data entry mistakes that occurred while the repository was being populated with product information.

Table 6.3. Summary table showing the number of design solutions found in both the student generated morphological matrices and the morphological matrices derived from the concept generator results.

		# of Solutions Generated	# of Solutions that Appear in Both Concept Generation Methods	% Overlap with Alternatively Produced Solutions
Power Supply Design Scenario	Student Generated Morphological Matrix with Solutions Classified Using Component Name Classifications	43	19	44.19%
	Automatically Generated Solutions Using Concept Generator Software	150	19	12.67%
Thermal Mug Design Scenario	Student Generated Morphological Matrix with Solutions Classified Using Component Name Classifications	79	35	44.30%
	Automatically Generated Solutions Using Concept Generator Software	152	35	23.03%
Wall Climber Toy Design Scenario	Student Generated Morphological Matrix with Solutions Classified Using Component Name Classifications	25	12	48.00%
	Automatically Generated Solutions Using Concept Generator Software	109	12	11.01%

6.2.2.2. Robustness Investigation Results. For the robustness evaluation activity, several function chains were selected from products previously dissected and analyzed by the undergraduate researchers, including a bug vacuum (a pest-removal device that utilizes a vacuum to trap bugs), an eyeglass cleaner, and a snow cone maker. Within the selected chains, components were swapped in a manner in which the chain still exhibited logical functionality. The original chain and the modified chain were then run through the concept generator software. An example of an original chain and its modified form from the bug vacuum is shown in Figure 6.8a. In each case, the top ranked conceptual solutions returned by each original chain input were also found highly ranked in the results returned by the modified chain input. Figure 6.8b shows the top 17 results for the original and modified chains in the bug vacuum example.




(b)

Concept Generator Results from Original Function Chain

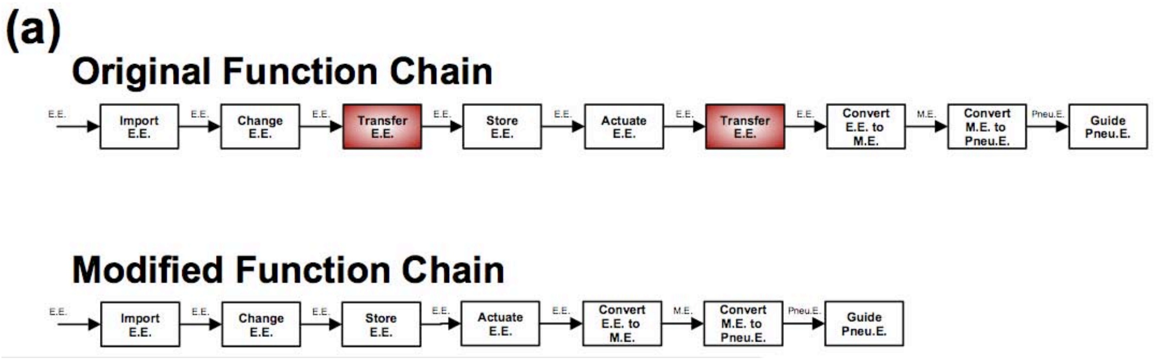
Rank	Component Chain →								
41454	Electric Cord	Electric Wire	Electric Wire	Battery	Switch	Electric Wire	Electric Motor	Fan	Tube
41409	Battery	Electric Wire	Electric Wire	Battery	Switch	Electric Wire	Electric Motor	Fan	Tube
41400	Electric Wire	Electric Wire	Electric Wire	Battery	Switch	Electric Wire	Electric Motor	Fan	Tube
41400	Electric Wire	Circuit Board	Electric Wire	Battery	Switch	Electric Wire	Electric Motor	Fan	Tube
41397	Switch	Electric Wire	Electric Wire	Battery	Switch	Electric Wire	Electric Motor	Fan	Tube
41397	Switch	Circuit Board	Electric Wire	Battery	Switch	Electric Wire	Electric Motor	Fan	Tube
41388	Circuit Board	Electric Wire	Electric Wire	Battery	Switch	Electric Wire	Electric Motor	Fan	Tube
41388	Circuit Board	Circuit Board	Electric Wire	Battery	Switch	Electric Wire	Electric Motor	Fan	Tube
41385	Housing	Electric Wire	Electric Wire	Battery	Switch	Electric Wire	Electric Motor	Fan	Tube
41385	Electric Motor	Electric Wire	Electric Wire	Battery	Switch	Electric Wire	Electric Motor	Fan	Tube
41385	Housing	Circuit Board	Electric Wire	Battery	Switch	Electric Wire	Electric Motor	Fan	Tube
41385	Electric Motor	Circuit Board	Electric Wire	Battery	Switch	Electric Wire	Electric Motor	Fan	Tube
41286	Electric Cord	Electric Wire	Electric Wire	Battery	Switch	Electric Wire	Electric Motor	Gear	?
41241	Battery	Electric Wire	Electric Wire	Battery	Switch	Electric Wire	Electric Motor	Gear	?
41238	Electric Cord	Electric Wire	Electric Wire	Battery	Switch	Electric Wire	Electric Motor	Seal	Tube
41236	Electric Cord	Electric Wire	Electric Wire	Battery	Switch	Electric Wire	Electric Motor	Seal	Valve
41234	Electric Cord	Electric Wire	Electric Wire	Battery	Switch	Electric Wire	Electric Motor	Driveshaft	?
41234	Electric Cord	Electric Wire	Electric Wire	Battery	Switch	Electric Wire	Electric Motor	Circuit Board	?
1232	Electric Wire	[illegible]	[illegible]	[illegible]	Switch	[illegible]	[illegible]	[illegible]	[illegible]

Concept Generator Results from Modified Function Chain

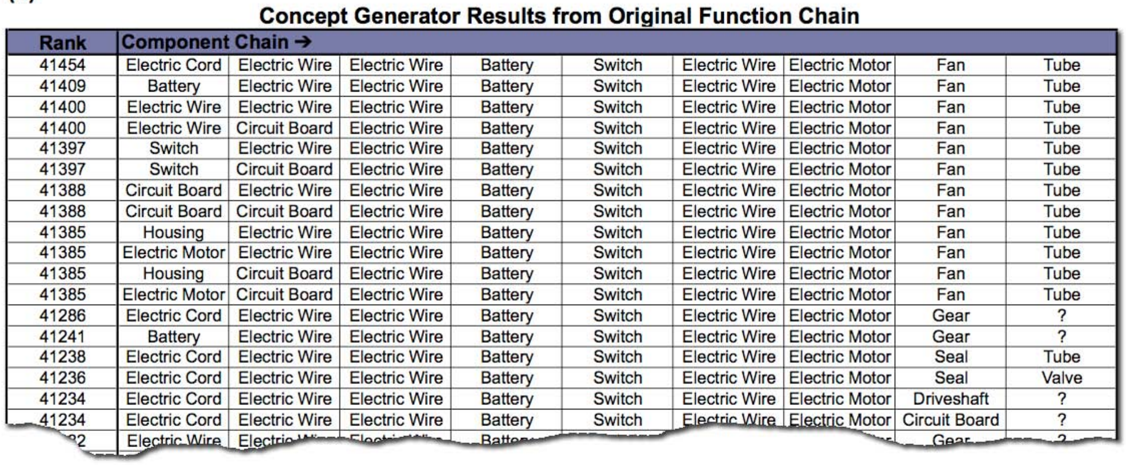
Rank	Component Chain →								
34630	Electric Cord	Electric Wire	Electric Wire	Battery	Electric Wire	Switch	Electric Motor	Fan	Tube
34462	Electric Cord	Electric Wire	Electric Wire	Battery	Electric Wire	Switch	Electric Motor	Gear	?
34414	Electric Cord	Electric Wire	Electric Wire	Battery	Electric Wire	Switch	Electric Motor	Seal	Tube
34412	Electric Cord	Electric Wire	Electric Wire	Battery	Electric Wire	Switch	Electric Motor	Seal	Valve
34410	Electric Cord	Electric Wire	Electric Wire	Battery	Electric Wire	Switch	Electric Motor	Driveshaft	?
34410	Electric Cord	Electric Wire	Electric Wire	Battery	Electric Wire	Switch	Electric Motor	Circuit Board	?
34360	Electric Cord	Electric Wire	Electric Wire	Battery	Electric Wire	Switch	Electric Motor	Housing	Tube
34359	Electric Cord	Electric Wire	Electric Wire	Battery	Electric Wire	Switch	Electric Motor	Switch	Valve
34359	Electric Cord	Electric Wire	Electric Wire	Battery	Electric Wire	Switch	Electric Motor	Housing	Valve
34358	Electric Cord	Electric Wire	Electric Wire	Battery	Electric Wire	Switch	Electric Motor	Link	?
31998	Electric Cord	Electric Wire	Electric Wire	Battery	Electric Wire	Switch	Switch	Gear	?
31990	Electric Cord	Electric Wire	Electric Wire	Battery	Electric Wire	Switch	Switch	Circuit Board	?
31990	Electric Cord	Electric Wire	Electric Wire	Battery	Electric Wire	Switch	Indicator Light	Circuit Board	?
31984	Electric Cord	Electric Wire	Electric Wire	Battery	Electric Wire	Switch	Switch	Housing	Tube
31984	Electric Cord	Electric Wire	Electric Wire	Battery	Electric Wire	Switch	Indicator Light	Housing	Tube
31983	Electric Cord	Electric Wire	Electric Wire	Battery	Electric Wire	Switch	Switch	Switch	Valve
31983	Electric Cord	Electric Wire	Electric Wire	Battery	Electric Wire	Switch	Switch	Housing	Valve
31983	Electric Cord	Electric Wire	Electric Wire	Battery	Electric Wire	Switch	[illegible]	Switch	Valve
31983	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

Figure 6.8. (a) Example function chain extracted from the full functional model of a bug vacuum both in the original form and permuted form. (b) The top concept generator results returned from the original and permuted chain shown above in (a).

6.2.2.3. Functional Model Variation Effects Results. Next, to investigate the effect that function omission has on the results returned by the concept generator, several function chains were selected from the students' pool of existing functional models that included functions that may be implicit in a designer-produced functional model. All functions that might not be explicitly included were then removed from the function chain, as shown in an example taken from the bug vacuum in Figure 6.9a. The original and the modified function chains were both run through the concept generator. The undergraduate researchers found that, for the chains entered, the modified function chains returned the same basic results as the original function chains. In the bug vacuum example shown in Figure 6.9b, the modified chain still generates concepts with the same major components as the original despite the removed functions, In addition, the students remarked that the number of concepts generated for the modified chain is much smaller and more manageable than the one generated for the original chain (195 concepts vs. 43136 concepts in the bug vacuum example shown); a situation that is expected given the combinatorial characteristics of assembling chains of solutions from . Additionally, the modified chain returned only complete concepts (in the example shown in Figure 6.9) while the original chain returned over 18,500 incomplete concepts.

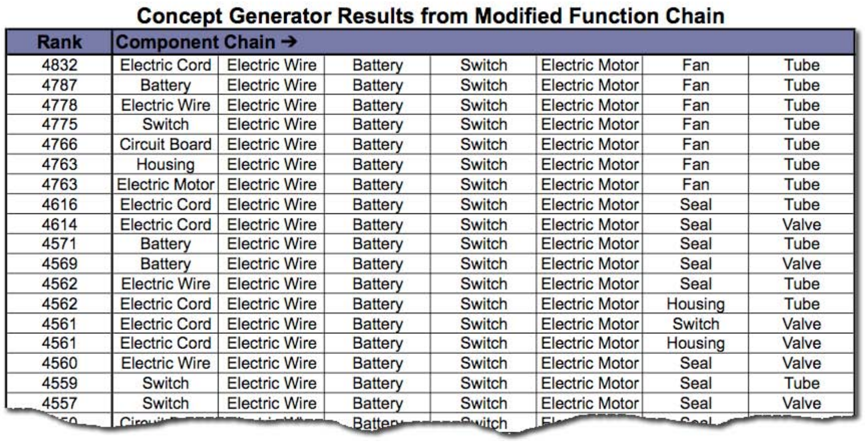


(b)



Concept Generator Results from Original Function Chain

Rank	Component Chain →								
41454	Electric Cord	Electric Wire	Electric Wire	Battery	Switch	Electric Wire	Electric Motor	Fan	Tube
41409	Battery	Electric Wire	Electric Wire	Battery	Switch	Electric Wire	Electric Motor	Fan	Tube
41400	Electric Wire	Electric Wire	Electric Wire	Battery	Switch	Electric Wire	Electric Motor	Fan	Tube
41400	Electric Wire	Circuit Board	Electric Wire	Battery	Switch	Electric Wire	Electric Motor	Fan	Tube
41397	Switch	Electric Wire	Electric Wire	Battery	Switch	Electric Wire	Electric Motor	Fan	Tube
41397	Switch	Circuit Board	Electric Wire	Battery	Switch	Electric Wire	Electric Motor	Fan	Tube
41388	Circuit Board	Electric Wire	Electric Wire	Battery	Switch	Electric Wire	Electric Motor	Fan	Tube
41388	Circuit Board	Circuit Board	Electric Wire	Battery	Switch	Electric Wire	Electric Motor	Fan	Tube
41385	Housing	Electric Wire	Electric Wire	Battery	Switch	Electric Wire	Electric Motor	Fan	Tube
41385	Electric Motor	Electric Wire	Electric Wire	Battery	Switch	Electric Wire	Electric Motor	Fan	Tube
41385	Housing	Circuit Board	Electric Wire	Battery	Switch	Electric Wire	Electric Motor	Fan	Tube
41385	Electric Motor	Circuit Board	Electric Wire	Battery	Switch	Electric Wire	Electric Motor	Fan	Tube
41286	Electric Cord	Electric Wire	Electric Wire	Battery	Switch	Electric Wire	Electric Motor	Gear	?
41241	Battery	Electric Wire	Electric Wire	Battery	Switch	Electric Wire	Electric Motor	Gear	?
41238	Electric Cord	Electric Wire	Electric Wire	Battery	Switch	Electric Wire	Electric Motor	Seal	Tube
41236	Electric Cord	Electric Wire	Electric Wire	Battery	Switch	Electric Wire	Electric Motor	Seal	Valve
41234	Electric Cord	Electric Wire	Electric Wire	Battery	Switch	Electric Wire	Electric Motor	Driveshaft	?
41234	Electric Cord	Electric Wire	Electric Wire	Battery	Switch	Electric Wire	Electric Motor	Circuit Board	?
[illegible]	Electric Wire	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]



Concept Generator Results from Modified Function Chain

Rank	Component Chain →						
4832	Electric Cord	Electric Wire	Battery	Switch	Electric Motor	Fan	Tube
4787	Battery	Electric Wire	Battery	Switch	Electric Motor	Fan	Tube
4778	Electric Wire	Electric Wire	Battery	Switch	Electric Motor	Fan	Tube
4775	Switch	Electric Wire	Battery	Switch	Electric Motor	Fan	Tube
4766	Circuit Board	Electric Wire	Battery	Switch	Electric Motor	Fan	Tube
4763	Housing	Electric Wire	Battery	Switch	Electric Motor	Fan	Tube
4763	Electric Motor	Electric Wire	Battery	Switch	Electric Motor	Fan	Tube
4616	Electric Cord	Electric Wire	Battery	Switch	Electric Motor	Seal	Tube
4614	Electric Cord	Electric Wire	Battery	Switch	Electric Motor	Seal	Valve
4571	Battery	Electric Wire	Battery	Switch	Electric Motor	Seal	Tube
4569	Battery	Electric Wire	Battery	Switch	Electric Motor	Seal	Valve
4562	Electric Wire	Electric Wire	Battery	Switch	Electric Motor	Seal	Tube
4562	Electric Cord	Electric Wire	Battery	Switch	Electric Motor	Housing	Tube
4561	Electric Cord	Electric Wire	Battery	Switch	Electric Motor	Switch	Valve
4561	Electric Cord	Electric Wire	Battery	Switch	Electric Motor	Housing	Valve
4560	Electric Wire	Electric Wire	Battery	Switch	Electric Motor	Seal	Valve
4559	Switch	Electric Wire	Battery	Switch	Electric Motor	Seal	Tube
4557	Switch	Electric Wire	Battery	Switch	Electric Motor	Seal	Valve
[illegible]	[illegible]	[illegible]	[illegible]	Switch	[illegible]	Seal	[illegible]

Figure 6.9. (a) Example function chain extracted from the full functional model of a bug vacuum both in the original form and with the assumed functionality omitted. (b) The top concept generator results returned from the original and modified chain shown above in (a).

6.3. EXPERIMENT: UNDERGRADUATE INVESTIGATION, PART II

To qualitatively evaluate the practicality of using the concept generators to produce conceptual design variants early in the design process, designers manually generated concepts and compared them against automatically generated design solutions for two design scenarios. The data collected during this comparison were later studied quantitatively, and the results can be found in Section 6.3.2. The chief objective of this study is to compare and analyze the concepts generated by hand versus those generated by the computer design synthesis tools in a 2^3 factorial design of experiment. The research participants were three undergraduate researchers from the University of Texas at Austin and University of Missouri-Rolla with roughly two to three years of college experience behind them. All of them have a basic understanding and experience with the design process including concept generation techniques. Throughout the experiment, each participant had access to a computer for documentation purposes. To avoid pollinating the manually generated morphological matrices with ideas from the design repository, the designers completed all manual concept generation tasks for each design scenario before moving on to generate design solutions using the concept generators. The timeline for the study spanned two weeks and was carried out as discussed in the following section.

6.3.1. Experimental Setup. The researchers were each presented two different design scenarios. In the first, the participants were asked to generate concepts by redesigning a drink mixer (Figure 6.10), a preexisting small kitchen appliance. Each researcher had the liberty to redesign the drink mixer without any specific customer needs to use as guidelines for the redesign process. In the second scenario, the participants were asked to design a bread slicer based on a given set of customer needs (see Table 6.4).

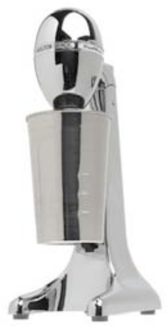


Figure 6.10. (Above) Original drink mixer design used during the redesign scenario.

Table 6.4. (Below) List of customer needs used for the original design of a bread slicer.

Sr. No.	Customer Need
1	Uniform slices
2	Accomodates any size loaf
3	Easy clean up (of crumbs)
4	Clean cut
5	Bread should not break / squish
6	Easy to use
7	Compact
8	Safe to use

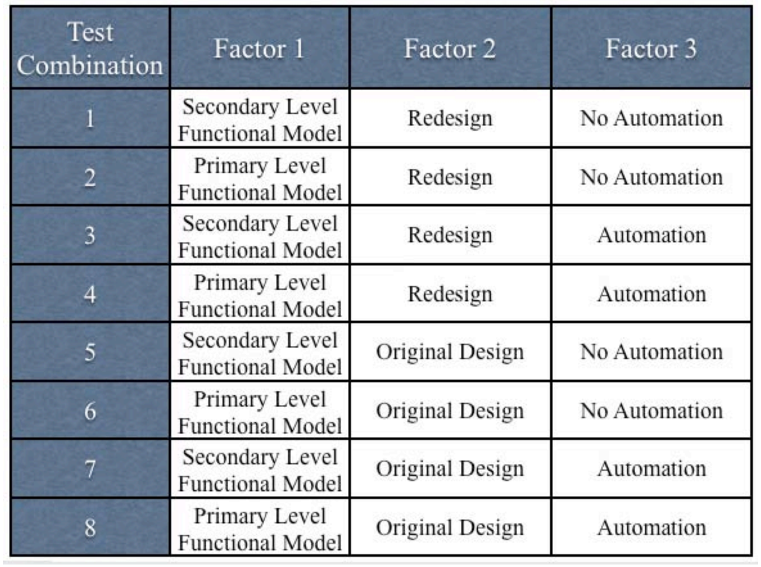
In both scenarios the functional models were generated using the primary and secondary Functional Basis as different start points for the concept generation exercise.

A full factorial experiment was carried out to test, study, and analyze the impact of these three factors on the concept generation.

- Beginning concept generation activities from a primary function structure versus a secondary function structure.
- Generating conceptual variants using automated tools versus manually brainstorming ideas.
- Producing conceptual variants for a redesign scenario versus an original design scenario.

A tabular summary of the three test factors and the eight factor combinations is shown below in Table 6.5.

Table 6.5. Summary of full factorial experimental test combinations performed by the research participants.



Test Combination	Factor 1	Factor 2	Factor 3
1	Secondary Level Functional Model	Redesign	No Automation
2	Primary Level Functional Model	Redesign	No Automation
3	Secondary Level Functional Model	Redesign	Automation
4	Primary Level Functional Model	Redesign	Automation
5	Secondary Level Functional Model	Original Design	No Automation
6	Primary Level Functional Model	Original Design	No Automation
7	Secondary Level Functional Model	Original Design	Automation
8	Primary Level Functional Model	Original Design	Automation

For test combinations 1 through 4, no customer needs were provided for the redesign of a drink mixer. The function models were generated by tearing down the drink mixer and understanding the basic functions exhibited by the existing product rather than by establishing the functional requirements from a set of customer needs. Hence, the starting point for the concept generation process was not a hypothetical functional model but the actual functional model of the product itself.

Each participant generated a functional model independently based on prior knowledge from the product teardown (see example shown in Figure 6.11) using function and flow terms strictly at the secondary level of the Functional Basis. A morphological matrix of conceptual solutions was next manually generated for each subfunction in the secondary level functional model (see Figure 6.12). Then, one brainstormed solution was picked for each subfunction, and a complete redesign solution was sketched for the drink mixer (see Figure 6.13).

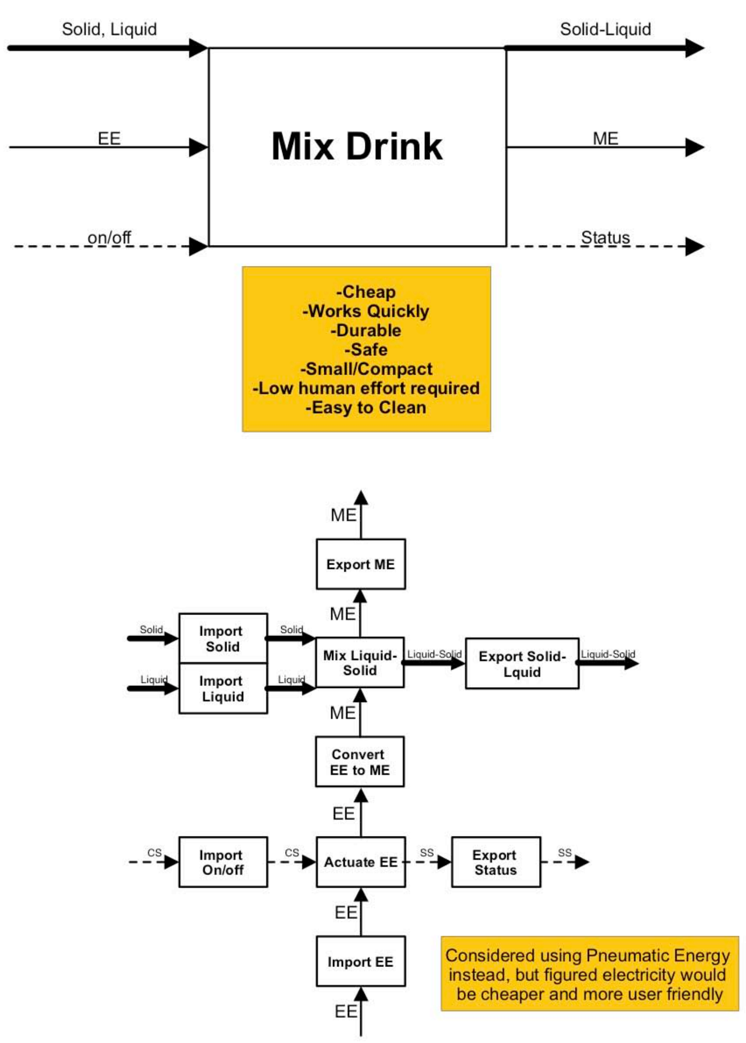
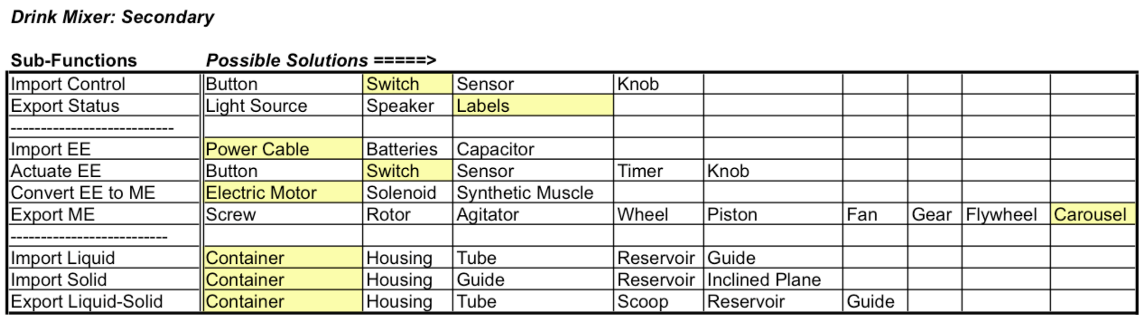


Figure 6.11. (Above) Functional model using the secondary level terms of the Functional Basis for the drink mixer redesign scenario.



Drink Mixer: Secondary

Sub-Functions	***Possible Solutions =====>***								
Import Control	Button	Switch	Sensor	Knob					
Export Status	Light Source	Speaker	Labels						

Import EE	Power Cable	Batteries	Capacitor						
Actuate EE	Button	Switch	Sensor	Timer	Knob				
Convert EE to ME	Electric Motor	Solenoid	Synthetic Muscle						
Export ME	Screw	Rotor	Agitator	Wheel	Piston	Fan	Gear	Flywheel	Carousel
-------------------------									
Import Liquid	Container	Housing	Tube	Reservoir	Guide				
Import Solid	Container	Housing	Guide	Reservoir	Inclined Plane				
Export Liquid-Solid	Container	Housing	Tube	Scoop	Reservoir	Guide			

Figure 6.12. Morphological matrix generated from the functional model shown in Figure 6.11 for the drink mixer redesign scenario.

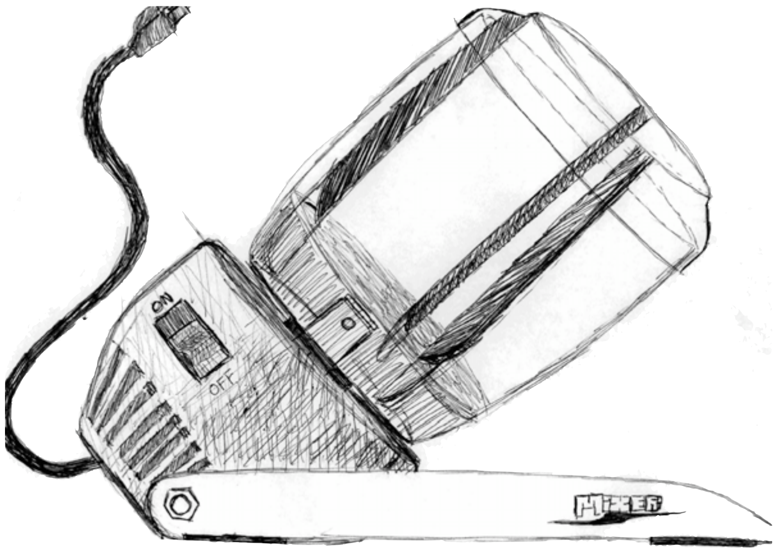
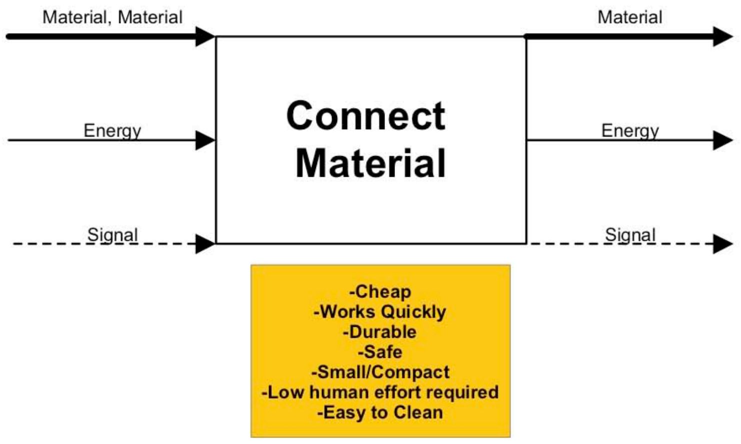


Figure 6.13. Solution sketch generated from the morphological matrix shown in Figure 6.12 for the drink mixer redesign scenario. Highlighted solutions shown in Figure 6.12 were used to produce this complete conceptual design.

Next, this process was repeated using only primary level function and flow terms from the Functional Basis to construct the functional model. Examples of a primary functional model, morphological matrix, and resulting embodiment sketch are shown in Figures 6.14-6.16.



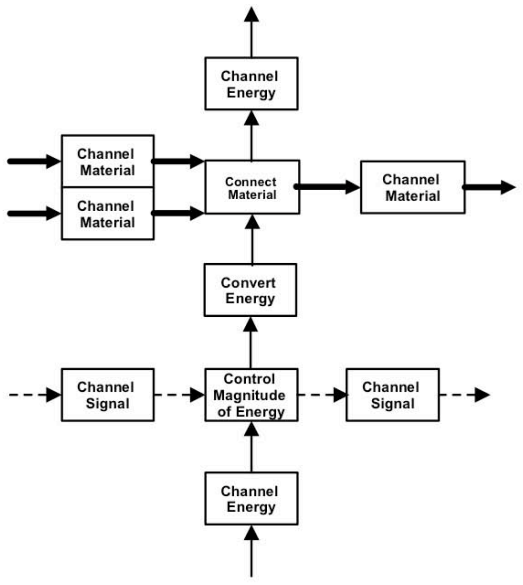
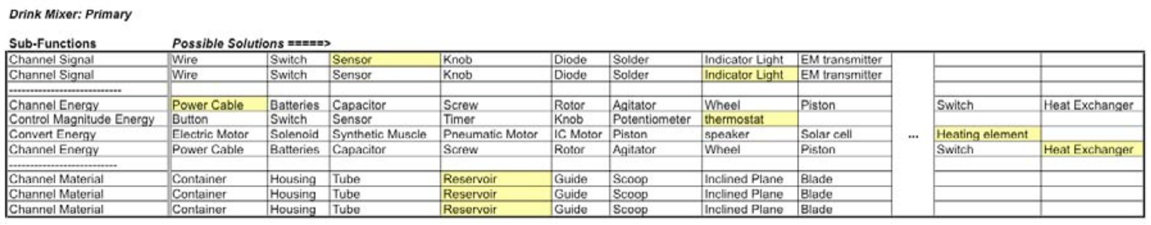


Figure 6.14. (Above) Functional model using the primary level terms of the Functional Basis for the drink mixer redesign scenario.



Drink Mixer: Primary

Sub-Functions	*Possible Solutions =====>*										
Channel Signal	Wire	Switch	Sensor	Knob	Diode	Solder	Indicator Light	EM transmitter			
Channel Signal	Wire	Switch	Sensor	Knob	Diode	Solder	Indicator Light	EM transmitter			

Channel Energy	Power Cable	Batteries	Capacitor	Screw	Rotor	Agitator	Wheel	Piston		Switch	Heat Exchanger
Control Magnitude Energy	Button	Switch	Sensor	Timer	Knob	Potentiometer	thermostat				
Convert Energy	Electric Motor	Solenoid	Synthetic Muscle	Pneumatic Motor	IC Motor	Piston	speaker	Solar cell	...	Heating element	
Channel Energy	Power Cable	Batteries	Capacitor	Screw	Rotor	Agitator	Wheel	Piston		Switch	Heat Exchanger
--------------------------											
Channel Material	Container	Housing	Tube	Reservoir	Guide	Scoop	Inclined Plane	Blade			
Channel Material	Container	Housing	Tube	Reservoir	Guide	Scoop	Inclined Plane	Blade			
Channel Material	Container	Housing	Tube	Reservoir	Guide	Scoop	Inclined Plane	Blade			

Figure 6.15. Morphological matrix generated from the functional model shown in Figure 6.14 for the drink mixer redesign scenario.

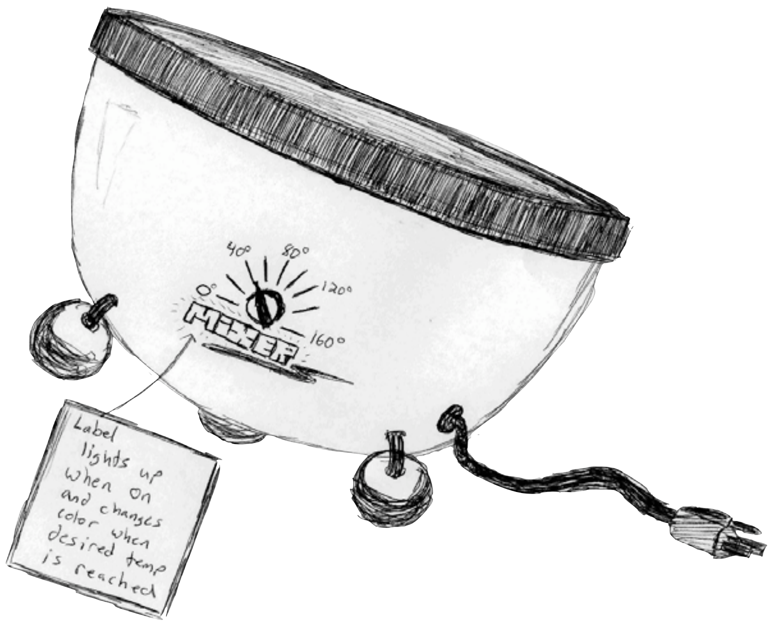
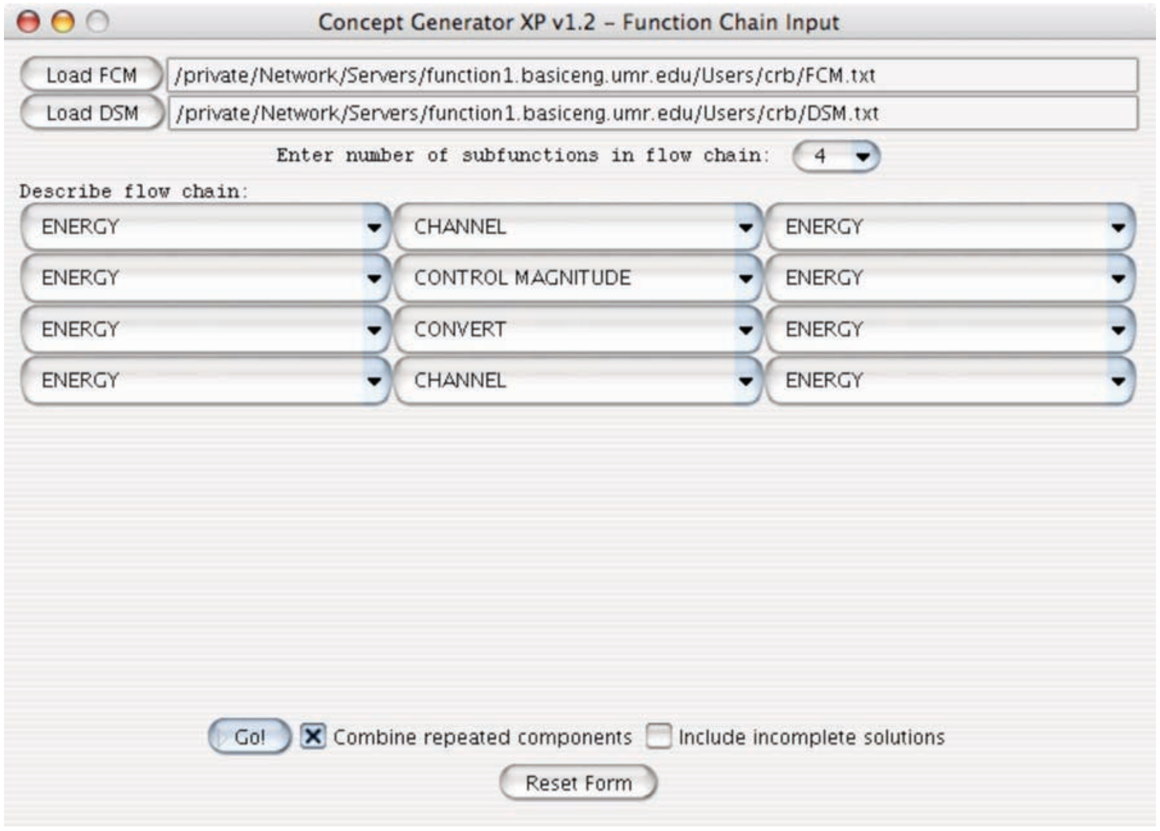

40°
80°
0°
120°
160°
MIXER
Label lights up when on and changes color when desired temp is reached

Figure 6.16. Solution sketch generated from the morphological matrix shown in Figure 6.15 for the drink mixer redesign scenario. Highlighted solutions shown in Figure 6.15 were used to produce this complete conceptual design.

Each of the two functional models created above (secondary and primary level) were then run through the concept generator to generate solutions. Sample solutions produced by the concept generator are shown in Figure 6.17.



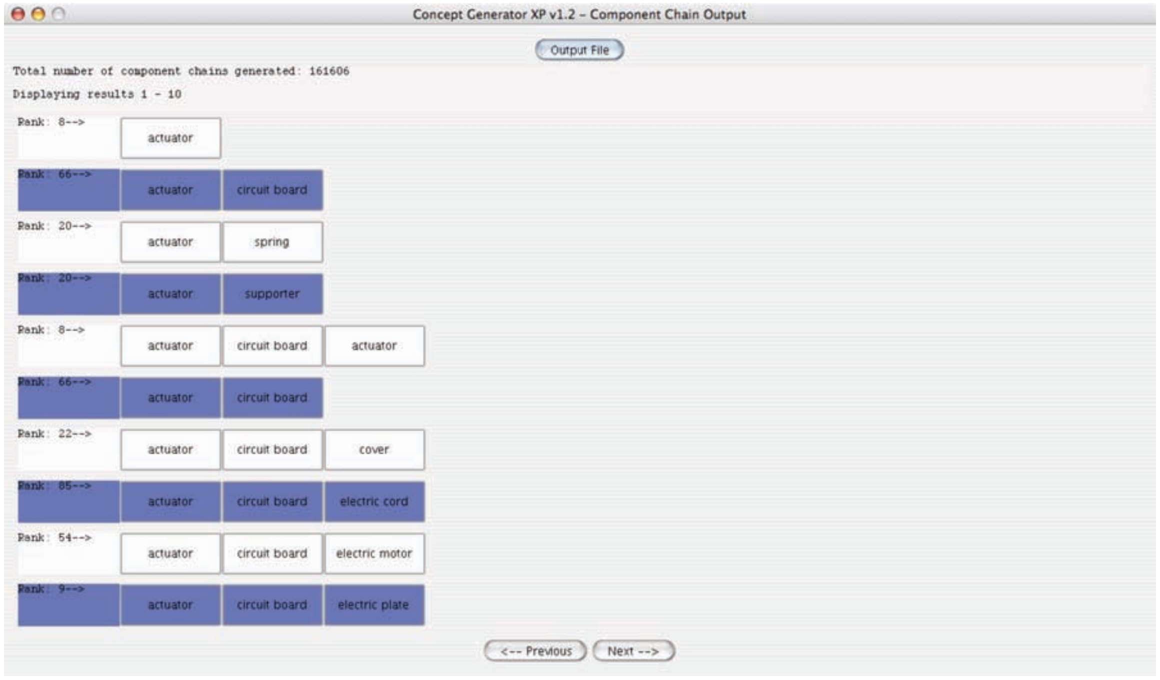
/private/Network/Servers/function1.basiceng.umr.edu/Users/crb/FCM.txt

/private/Network/Servers/function1.basiceng.umr.edu/Users/crb/DSM.txt

```
Enter number of subfunctions in flow chain: 4
Describe flow chain:
```

ENERGY	CHANNEL	ENERGY
ENERGY	CONTROL MAGNITUDE	ENERGY
ENERGY	CONVERT	ENERGY
ENERGY	CHANNEL	ENERGY

[✓] Combine repeated components [ ] Include incomplete solutions



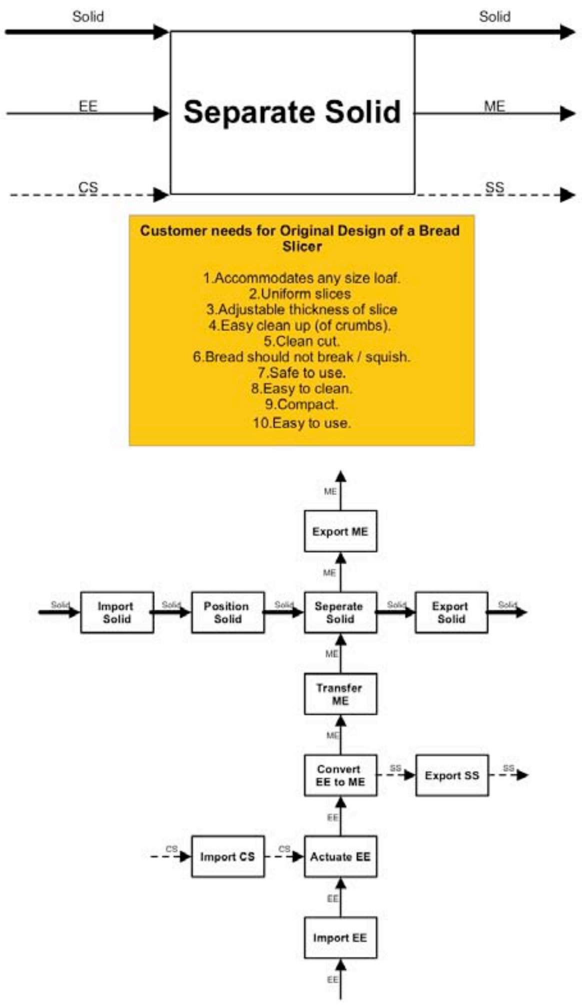
```
Total number of component chains generated: 161606
Displaying results 1 - 10
```

Rank: 8-->	actuator		
Rank: 66-->	actuator	circuit board	
Rank: 20-->	actuator	spring	
Rank: 20-->	actuator	supporter	
Rank: 8-->	actuator	circuit board	actuator
Rank: 66-->	actuator	circuit board	
Rank: 22-->	actuator	circuit board	cover
Rank: 85-->	actuator	circuit board	electric cord
Rank: 54-->	actuator	circuit board	electric motor
Rank: 9-->	actuator	circuit board	electric plate

Figure 6.17. Sample matrix-based concept generator output for a primary level functional model input for the drink mixer redesign.

The same experimental procedure was used for the original design of a bread slicer with one minor change. For this original design scenario, the designers began by producing functional models (secondary and primary) for each product from customer needs using the design steps previously shown in Figure 6.2.

As before, the three researchers independently developed primary and secondary level functional models from which to develop conceptual solutions. Since the researchers used no prior knowledge of existing products, the functional models generated were purely conceptual. Concepts were generated from each of the two functional models with and without automation. Sample results from this design scenario are shown below in Figures 6.18-6.23.



Customer needs for Original Design of a Bread Slicer

1.Accommodates any size loaf.
2.Uniform slices
3.Adjustable thickness of slice
4.Easy clean up (of crumbs).
5.Clean cut.
6.Bread should not break / squish.
7.Safe to use.
8.Easy to clean.
9.Compact.
10.Easy to use.

Figure 6.18. (Above) Functional model using the primary level terms of the Functional Basis for the bread slicer original design scenario.

Bread Slicer: Secondary Level

Sub-function:	***Possible Solutions =======>***							
Import Solid	Housing	Reservoir	Container	Carousel	Door	Scoop	Support	Human
Position Solid	Stop	Markings	Human					
Export Solid	Housing	Reservoir	Container	Carousel	Door	Scoop	Support	Human
--------------------								
Import CS	Switch	Sensor	Chip	circuit board	button			
Export CS	Markings	indicator light	speaker					

Import EE	Electric Cord	Battery	Capacitor					
Actuate EE	Switch	Sensor	Button					
Convert EE to ME	Electric Motor	Actuator	Synthetic Muscle					
Transfer ME	Link	Pulley	Belt	Gear	Shaft	Cable	extension	Cam
Export ME	Cable	Shaft	Blade	needle	wheel			

Figure 6.19. Morphological matrix generated from the functional model shown in Figure 6.18 for the bread slicer original design scenario.

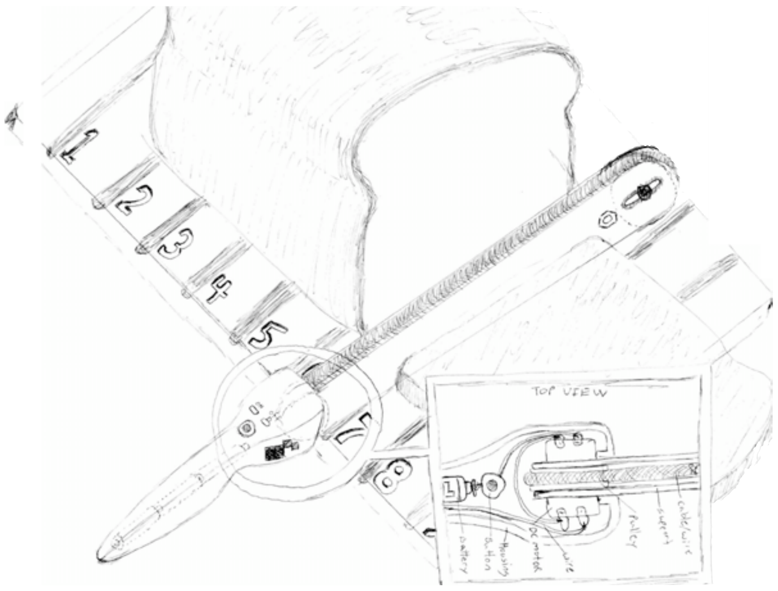
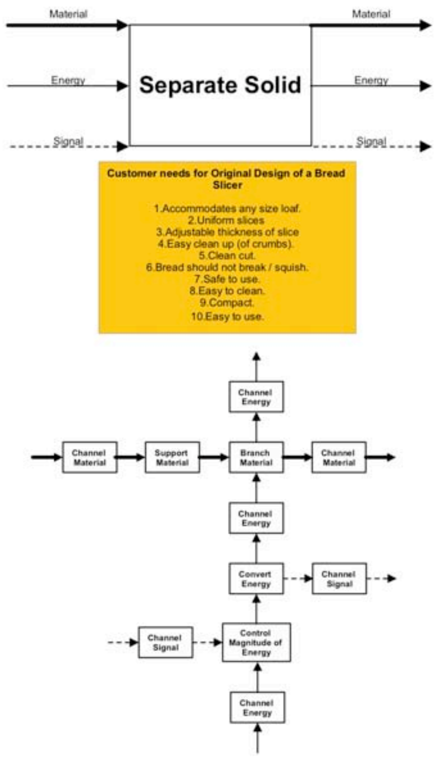


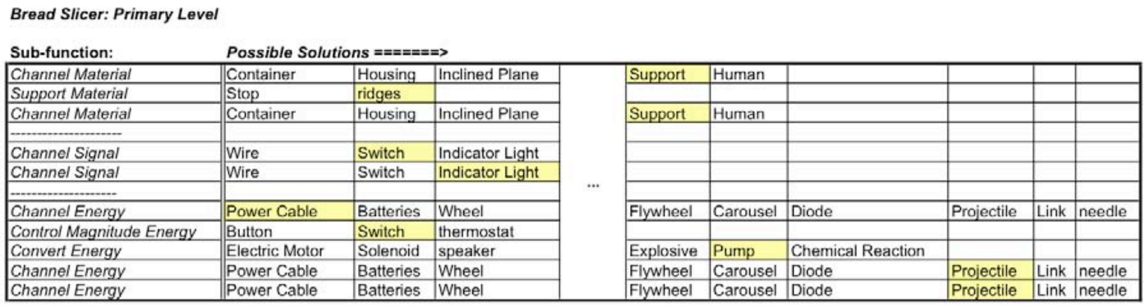
Figure 6.20. (Above) Solution sketch generated from the morphological matrix shown in Figure 6.19 for the bread slicer original design scenario. Highlighted solutions shown in Figure 6.19 were used to produce this complete conceptual design.



Customer needs for Original Design of a Bread Slicer

1.Accommodates any size loaf.
2.Uniform slices
3.Adjustable thickness of slice
4.Easy clean up (of crumbs).
5.Clean cut.
6.Bread should not break / squish.
7.Safe to use.
8.Easy to clean.
9.Compact.
10.Easy to use.

Figure 6.21. Functional model using the primary level terms of the Functional Basis for the bread slicer original design scenario.



Bread Slicer: Primary Level

Sub-function:	Possible Solutions =======>									
Channel Material	Container	Housing	Inclined Plane		Support	Human				
Support Material	Stop	ridges								
Channel Material	Container	Housing	Inclined Plane		Support	Human				

Channel Signal	Wire	Switch	Indicator Light							
Channel Signal	Wire	Switch	Indicator Light	...						

Channel Energy	Power Cable	Batteries	Wheel		Flywheel	Carousel	Diode	Projectile	Link	needle
Control Magnitude Energy	Button	Switch	thermostat							
Convert Energy	Electric Motor	Solenoid	speaker		Explosive	Pump	Chemical Reaction			
Channel Energy	Power Cable	Batteries	Wheel		Flywheel	Carousel	Diode	Projectile	Link	needle
Channel Energy	Power Cable	Batteries	Wheel		Flywheel	Carousel	Diode	Projectile	Link	needle

Figure 6.22. (Above) Morphological matrix generated from the functional model shown in Figure 6.21 for the bread slicer original design scenario.

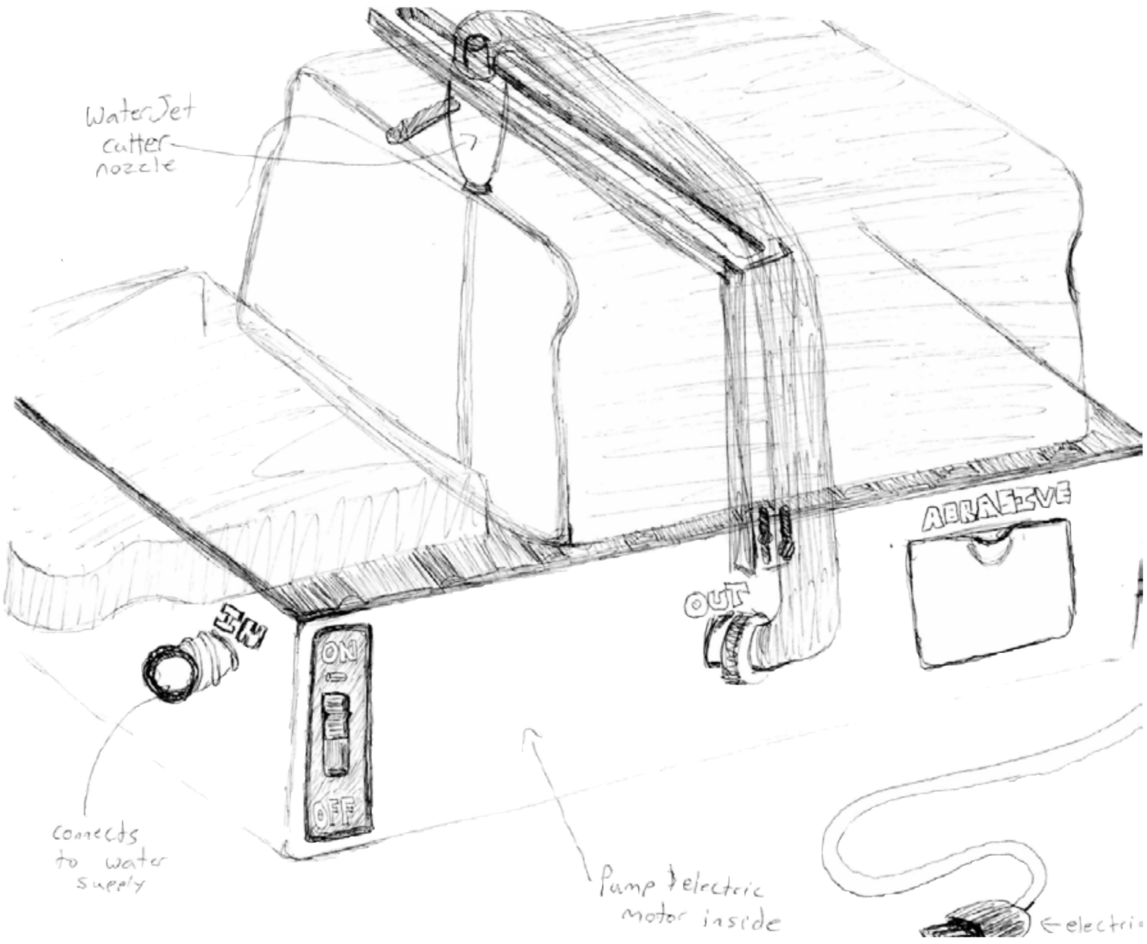


Figure 6.23. Solution sketch generated from the morphological matrix shown in Figure 6.22 for the bread slicer original design scenario. Highlighted solutions shown in Figure 6.22 were used to produce this complete conceptual design.

6.3.2. Results. The following sections present the results from the qualitative comparisons as well as the post-investigation quantitative study. Results from the designers' evaluation activities indicated that, in general, both the primary and secondary modeling levels are useful for modeling a product's functional requirements under different design scenarios. In general, primary level functional models are more abstract and increase creativity in design and secondary level models increase a designer's focus and speed at converging on a feasible design. In addition, with a few notable exceptions, the matrix-based concept generator consistently averaged a larger quantity of feasible solutions for each subfunction than those produced manually by the students. In general, the research participants felt the matrix-based concept generator helped stimulate new creative solutions to the design problems given, but noted that the number of concepts returned from primary level functional inputs produced was largely unmanageable and would benefit greatly from additional constraints and filters to eliminate concepts that were largely irrelevant to the specific design problem.

6.3.2.1. Qualitative Analysis. The concepts generated were evaluated by the undergraduate researchers in terms of value to them during the design process. This evaluation primarily involved qualitatively analyzing each combination of test factors and determining the combinations of factors (i.e. primary vs. secondary level of functional modeling, manual vs. automated concept generation) produced more diverse sets of concepts when used to develop concepts for the redesign and original design scenarios. Additionally, similarities among the concepts produced were also examined. Seeking the existing design embodiment for the drink mixer as well as the brainstormed solutions for each scenario among the concepts generated by the automated tools may help elucidate the comprehensiveness, feasibility, and novelty of the generated variants. Key questions asked during the qualitative evaluations include:

(1) Which level of model detail (primary vs. secondary) is more valuable to a designer during the process of (re)designing a product?

(2) Do any of the solutions returned by the concept generator at the primary/ secondary level of detail give ideas for solutions that were not achieved by manual methods?

(3) Which level of model detail (primary vs. secondary) returns results from the automated concept generators that are more useful to a designer during the process of (re) designing a product?

During the drink mixer redesign activities, the research participants observed that the primary level functional model allowed for greater freedom to generate a wide assortment of conceptual solutions, both manually and by using the automated concept generators. By including more abstract terms to describe the functional requirements, primary functional models help to broaden the solution space and enhance creativity and novelty without eliminating the ideas similar to the existing design. However, many of the solutions generated computationally from the primary level functional models by the matrix-based concept generation method were not feasible or relevant to the given design objective. In contrast, the concepts generated from the secondary function model were complete and practical but tended to be too specific and too similar to the original existing design to stimulate new creative solutions by the research participants. Hence, these results lead to the conclusion that the primary level of functional model is more valuable if a complete revision of the existing product with the same functionality is desired, whereas the secondary level functional model will be more beneficial if only minor revisions to the configuration of components are sought without making changes across solution domains. For maximum versatility, the choice of a secondary or primary functional model should be left open to the designer.

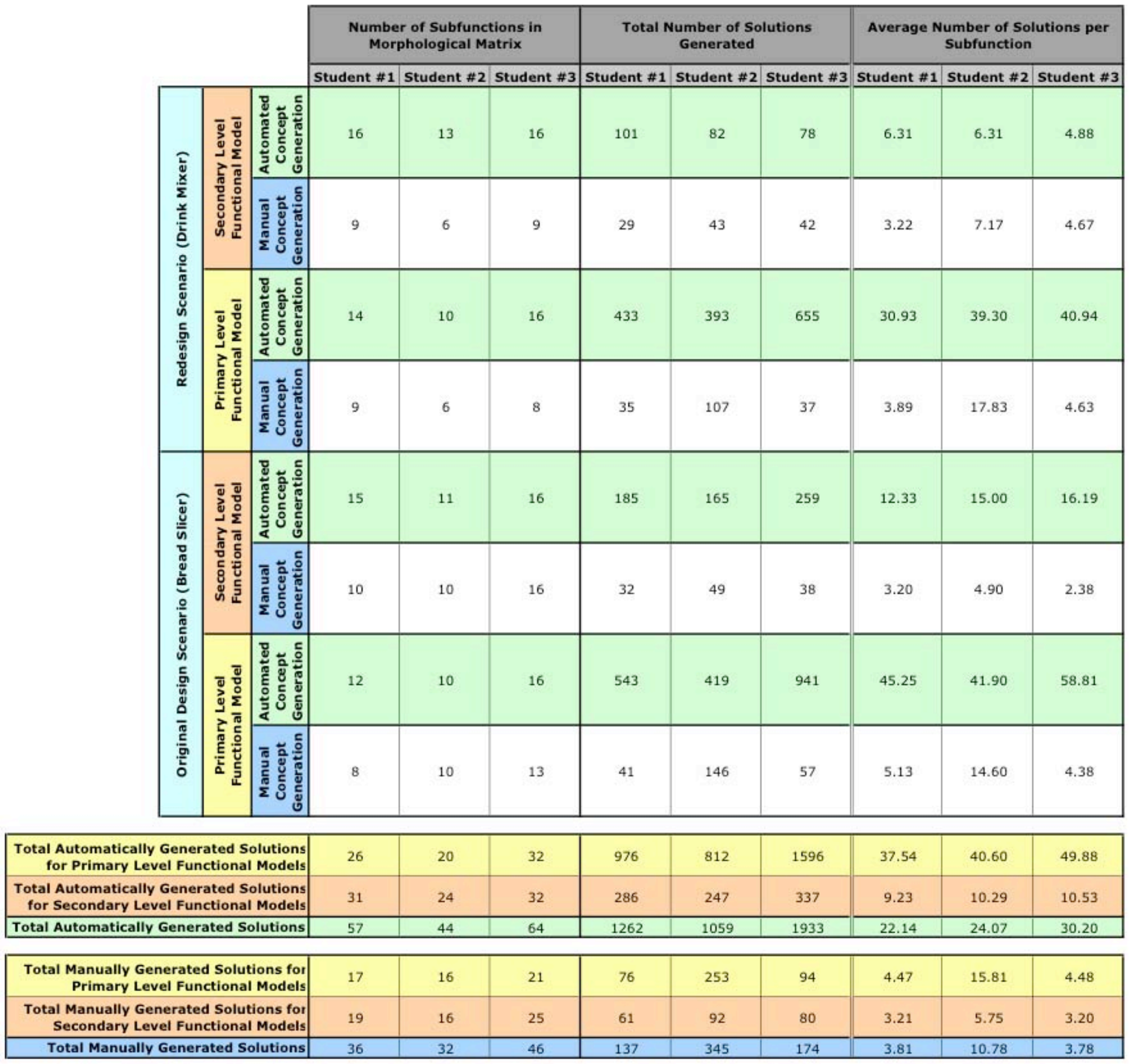
In the bread slicer original design scenario, the concepts generated from the primary level were very diverse. The matrix-based concept generator generated numerous concepts. However, the participants reported difficulty in parsing through the returned concepts and narrowing down the results without any additional constraints beyond the given set of customer needs. For the bread slider design, the secondary level functional model generated more complete and feasible solutions, but, in general, the concepts generated tended to be less creative when compared to those generated by the primary functional model.

The research participants observed a critical need to filter out obscure and absurd solutions (with respect to the specific design needs for the product to be designed) if the designer is to proceed through the design process using the primary level functional model, especially when the matrix-based method of automated concept generation is employed. This filtering of solutions is complicated and is subjective to say the least, but one proposed way to help create design boundaries for the results would be to combine the primary and secondary level Functional Basis terms within a single functional model. Design functions and flow domains that the designer is certain that the product needs to include may be expressed at the more concrete secondary level of language, while the primary level of language may be used to designate the less well-defined areas of the design. This combination of detail level is anticipated to help eliminate some of the less applicable design while still encouraging a broad array of solution exploration.

6.3.2.2. Quantitative Analysis. As an extension to the qualitative comparison performed by the undergraduate researchers, the manual design solutions generated by the students during each design scenario were later compared quantitatively to those generated by the matrix-based concept generator from the design repository knowledge. Since the student design solutions were contained in a morphological matrix while the set of computationally generated solutions consisted of lists of compatible component chains, direct comparisons of the solutions were difficult to achieve. In order to make quantitative comparisons, the results returned by the matrix-based concept generator were separated into morphological matrices that could then be directly compared to the morphological matrices manually generated by the students. The computationally derived morphological matrices were produced by dissociating each component solution from each chain of compatible components and recording the unique solutions produced for each function entered.

The functional models developed during the design process were divided into single non-branching chains of functions and each chain was entered into the matrix-based concept generator software. In the case of the drink mixer design, for instance, the full conceptual functional model shown in Figure 6.11 was broken into three function chains. The lists of design solutions produced by the matrix-based concept generator were saved as text files. Once the software generated design solutions had been distilled into morphological matrices, numerous observations could be made regarding the quantity of solutions produced by each method. Looking at the total number of distinct design solutions generated during each test combination, the matrix-based concept generator produced more design solutions per subfunction than the students produced manually (22.14 vs. 3.81 for student #1, 24.07 vs. 10.78 for student #2, and 30.20 vs. 3.78 for student #3, as shown in Table 6.6). For each subfunction, solutions were translated to the component naming scheme and distinct solutions refer to the number of different component names plus any additional solutions that were not classifiable.

Table 6.6. Summary table showing the total number of solutions for the morphological matrix from each testing combination. The number of subfunctions included in each morphological matrix and the average number of solutions per subfunction are also shown.

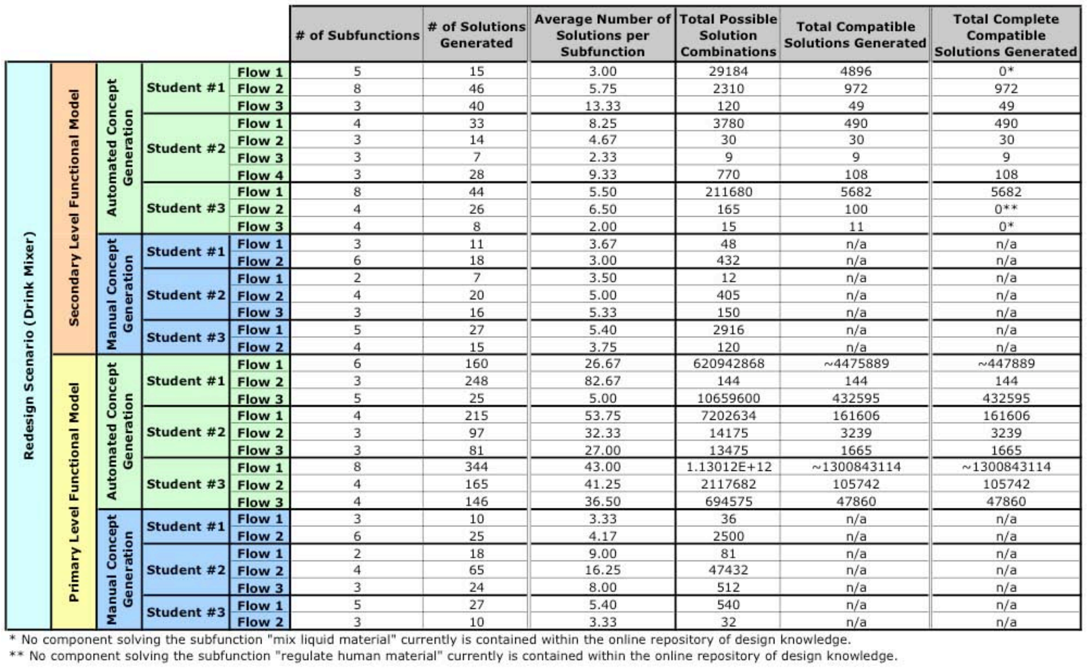


			Number of Subfunctions in Morphological Matrix			Total Number of Solutions Generated			Average Number of Solutions per Subfunction		
			Student #1	Student #2	Student #3	Student #1	Student #2	Student #3	Student #1	Student #2	Student #3
Redesign Scenario (Drink Mixer)	Secondary Level Functional Model	Automated Concept Generation	16	13	16	101	82	78	6.31	6.31	4.88
		Manual Concept Generation	9	6	9	29	43	42	3.22	7.17	4.67
	Primary Level Functional Model	Automated Concept Generation	14	10	16	433	393	655	30.93	39.30	40.94
		Manual Concept Generation	9	6	8	35	107	37	3.89	17.83	4.63
Original Design Scenario (Bread Slicer)	Secondary Level Functional Model	Automated Concept Generation	15	11	16	185	165	259	12.33	15.00	16.19
		Manual Concept Generation	10	10	16	32	49	38	3.20	4.90	2.38
	Primary Level Functional Model	Automated Concept Generation	12	10	16	543	419	941	45.25	41.90	58.81
		Manual Concept Generation	8	10	13	41	146	57	5.13	14.60	4.38
Total Automatically Generated Solutions for Primary Level Functional Models			26	20	32	976	812	1596	37.54	40.60	49.88
Total Automatically Generated Solutions for Secondary Level Functional Models			31	24	32	286	247	337	9.23	10.29	10.53
Total Automatically Generated Solutions			57	44	64	1262	1059	1933	22.14	24.07	30.20
Total Manually Generated Solutions for Primary Level Functional Models			17	16	21	76	253	94	4.47	15.81	4.48
Total Manually Generated Solutions for Secondary Level Functional Models			19	16	25	61	92	80	3.21	5.75	3.20
Total Manually Generated Solutions			36	32	46	137	345	174	3.81	10.78	3.78

Table 6.7 gives a more detailed breakdown of the number of solutions and feasible solution chains produced by each method for each design scenario. Data within these tables are organized by the flow chains that were entered into the matrix-based concept generator to produce corresponding chains of compatible partial solutions. These

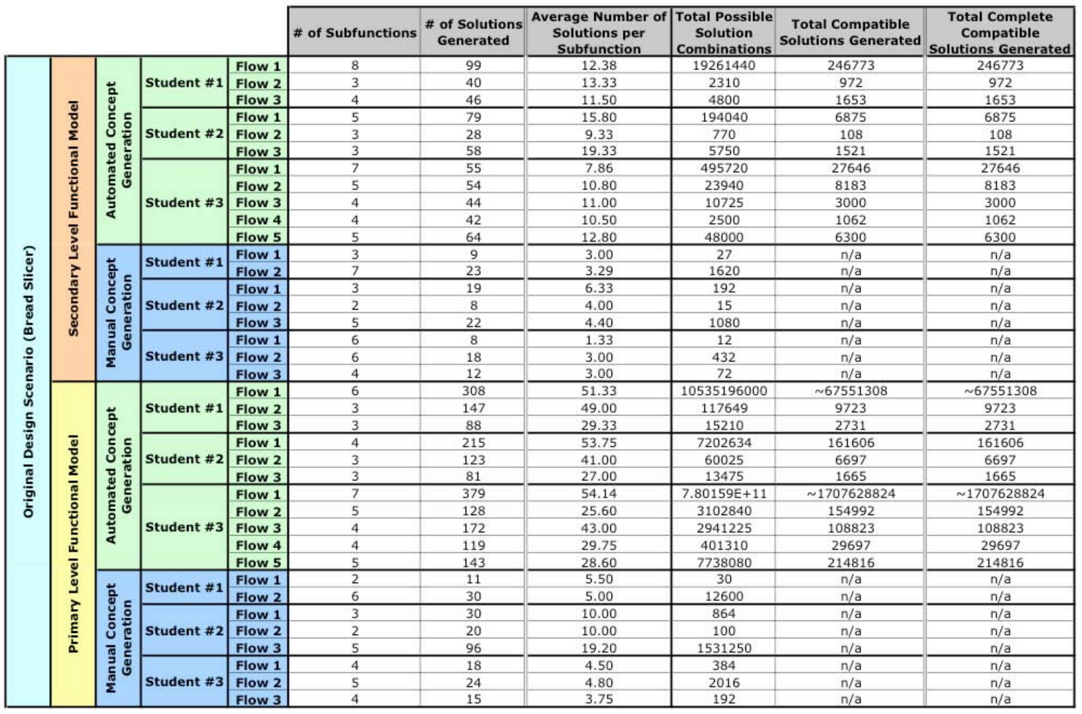
tables show that the average number of solutions produced per subfunction for nearly every flow is higher for the matrix-based concept generator group vs. the student generated groups of solutions. Correspondingly, the total number of complete compatible solutions produced by the matrix-based concept generator from the repository of design knowledge is typically greater than the total number of possible combinations produced by the students, with a few notable exceptions. First, in Table 6.7a, we can see that no complete solutions were assembled by the matrix-based concept generator for “Flow 1” under Student #1 or “Flow 3” under Student #3 in the drink mixer redesign scenario. This observation stems from the fact that, at this time, no component in the design repository solves the subfunction “mix liquid material”. Similarly, in Table 6.7a, the lack of solutions for “Flow 2” under Student #3 for the drink mixer redesign scenario results from the matrix-based concept generator being unable to find a component solution to the subfunction “regulate human material.

Table 6.7. Summary tables showing the # of subfunctions in each flow extracted from each functional model, total # of solutions generated for all of the subfunctions in each flow, avg. # of solutions per subfunction within a flow, # of compatible solution chains able to be constructed (both partial and complete) to solve the flow, and the total # of solution chain combinations possible (both feasible and infeasible) for the (a) drink mixer redesign, and (b) bread slicer original design scenarios.



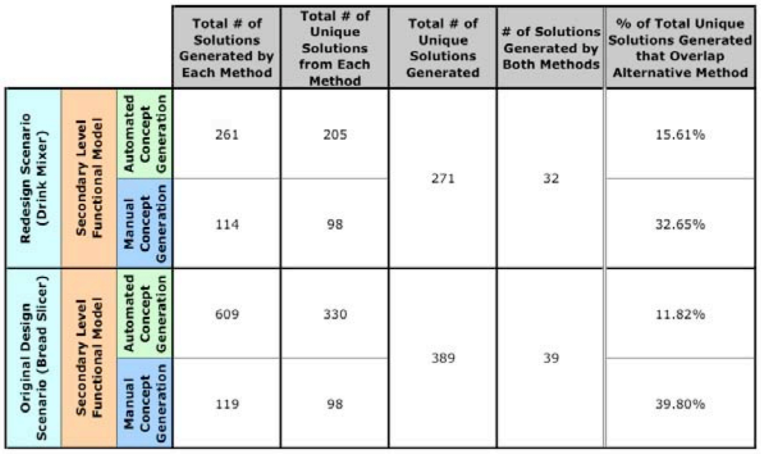
					# of Subfunctions	# of Solutions Generated	Average Number of Solutions per Subfunction	Total Possible Solution Combinations	Total Compatible Solutions Generated	Total Complete Compatible Solutions Generated
Redesign Scenario (Drink Mixer)	Secondary Level Functional Model	Automated Concept Generation	Student #1	Flow 1	5	15	3.00	29184	4896	0*
				Flow 2	8	46	5.75	2310	972	972
				Flow 3	3	40	13.33	120	49	49
			Student #2	Flow 1	4	33	8.25	3780	490	490
				Flow 2	3	14	4.67	30	30	30
				Flow 3	3	7	2.33	9	9	9
				Flow 4	3	28	9.33	770	108	108
			Student #3	Flow 1	8	44	5.50	211680	5682	5682
				Flow 2	4	26	6.50	165	100	0**
				Flow 3	4	8	2.00	15	11	0*
		Manual Concept Generation	Student #1	Flow 1	3	11	3.67	48	n/a	n/a
				Flow 2	6	18	3.00	432	n/a	n/a
			Student #2	Flow 1	2	7	3.50	12	n/a	n/a
				Flow 2	4	20	5.00	405	n/a	n/a
				Flow 3	3	16	5.33	150	n/a	n/a
			Student #3	Flow 1	5	27	5.40	2916	n/a	n/a
				Flow 2	4	15	3.75	120	n/a	n/a
	Primary Level Functional Model	Automated Concept Generation	Student #1	Flow 1	6	160	26.67	620942868	~4475889	~447889
				Flow 2	3	248	82.67	144	144	144
				Flow 3	5	25	5.00	10659600	432595	432595
			Student #2	Flow 1	4	215	53.75	7202634	161606	161606
				Flow 2	3	97	32.33	14175	3239	3239
				Flow 3	3	81	27.00	13475	1665	1665
			Student #3	Flow 1	8	344	43.00	1.13012E+12	~1300843114	~1300843114
				Flow 2	4	165	41.25	2117682	105742	105742
				Flow 3	4	146	36.50	694575	47860	47860
		Manual Concept Generation	Student #1	Flow 1	3	10	3.33	36	n/a	n/a
				Flow 2	6	25	4.17	2500	n/a	n/a
			Student #2	Flow 1	2	18	9.00	81	n/a	n/a
				Flow 2	4	65	16.25	47432	n/a	n/a
				Flow 3	3	24	8.00	512	n/a	n/a
			Student #3	Flow 1	5	27	5.40	540	n/a	n/a
				Flow 2	3	10	3.33	32	n/a	n/a

* No component solving the subfunction "mix liquid material" currently is contained within the online repository of design knowledge.
** No component solving the subfunction "regulate human material" currently is contained within the online repository of design knowledge.



					# of Subfunctions	# of Solutions Generated	Average Number of Solutions per Subfunction	Total Possible Solution Combinations	Total Compatible Solutions Generated	Total Complete Compatible Solutions Generated
Original Design Scenario (Bread Slicer)	Secondary Level Functional Model	Automated Concept Generation	Student #1	Flow 1	8	99	12.38	19261440	246773	246773
				Flow 2	3	40	13.33	2310	972	972
				Flow 3	4	46	11.50	4800	1653	1653
			Student #2	Flow 1	5	79	15.80	194040	6875	6875
				Flow 2	3	28	9.33	770	108	108
				Flow 3	3	58	19.33	5750	1521	1521
			Student #3	Flow 1	7	55	7.86	495720	27646	27646
				Flow 2	5	54	10.80	23940	8183	8183
				Flow 3	4	44	11.00	10725	3000	3000
				Flow 4	4	42	10.50	2500	1062	1062
				Flow 5	5	64	12.80	48000	6300	6300
		Manual Concept Generation	Student #1	Flow 1	3	9	3.00	27	n/a	n/a
				Flow 2	7	23	3.29	1620	n/a	n/a
			Student #2	Flow 1	3	19	6.33	192	n/a	n/a
				Flow 2	2	8	4.00	15	n/a	n/a
				Flow 3	5	22	4.40	1080	n/a	n/a
			Student #3	Flow 1	6	8	1.33	12	n/a	n/a
				Flow 2	6	18	3.00	432	n/a	n/a
				Flow 3	4	12	3.00	72	n/a	n/a
	Primary Level Functional Model	Automated Concept Generation	Student #1	Flow 1	6	308	51.33	10535196000	~67551308	~67551308
				Flow 2	3	147	49.00	117649	9723	9723
				Flow 3	3	88	29.33	15210	2731	2731
			Student #2	Flow 1	4	215	53.75	7202634	161606	161606
				Flow 2	3	123	41.00	60025	6697	6697
				Flow 3	3	81	27.00	13475	1665	1665
			Student #3	Flow 1	7	379	54.14	7.80159E+11	~1707628824	~1707628824
				Flow 2	5	128	25.60	3102840	154992	154992
				Flow 3	4	172	43.00	2941225	108823	108823
				Flow 4	4	119	29.75	401310	29697	29697
				Flow 5	5	143	28.60	7738080	214816	214816
		Manual Concept Generation	Student #1	Flow 1	2	11	5.50	30	n/a	n/a
				Flow 2	6	30	5.00	12600	n/a	n/a
			Student #2	Flow 1	3	30	10.00	864	n/a	n/a
				Flow 2	2	20	10.00	100	n/a	n/a
				Flow 3	5	96	19.20	1531250	n/a	n/a
			Student #3	Flow 1	4	18	4.50	384	n/a	n/a
				Flow 2	5	24	4.80	2016	n/a	n/a
				Flow 3	4	15	3.75	192	n/a	n/a

Since quantity of results is not the only concern when evaluating the usability of a design tool in concept generation, a comparison of the type of solutions produced by the matrix-based concept generator against those produced by the students was also made. Table 6.8 shows a summary of the number of overlapping design solutions seen in both the student generated and computationally derived morphological matrices. For instance, for the drink mixer data, of the 271 distinct solutions produced by the matrix-based concept generator plus all of the three designers, 32 of the solutions were generated by both of the two methods. This means that 32.65% of the designer-generated solutions were contained within the automatically generated solution set, and, alternatively, 15.61% of the concept generator results were contained within the designer-generated set of solutions.



			Total # of Solutions Generated by Each Method	Total # of Unique Solutions from Each Method	Total # of Unique Solutions Generated	# of Solutions Generated by Both Methods	% of Total Unique Solutions Generated that Overlap Alternative Method
Redesign Scenario (Drink Mixer)	Secondary Level Functional Model	Automated Concept Generation	261	205	271	32	15.61%
		Manual Concept Generation	114	98			32.65%
Original Design Scenario (Bread Slicer)	Secondary Level Functional Model	Automated Concept Generation	609	330	389	39	11.82%
		Manual Concept Generation	119	98			39.80%

Table 6.8. Summary table showing the number of distinct design solutions found in both the student generated morphological matrices and the morphological matrices derived from the matrix-based concept generator results.

6.4. CASE STUDY: A DOG FOOD PACKET COUNTER

In this section, both of the previously described methods of automated concept generation (the list-based output method and the interactive morphological matrix method described in Section 4.2 and 4.4, respectively) along with the web-based morphological search, described in Section 2.4.3., are evaluated using a design problem to transform an imprecise counting and packaging line at the Rolla Area Sheltered Workshop. The solutions generated for that design problem are used here to compare the results of manual concept generation techniques with the results from each of the three automated methods. The device, prototyped at the University of Missouri–Rolla (UMR), was the product of several modern design methodologies. Initial customer interviews were conducted, a customer needs questionnaire was developed, technical requirements were formed, and several methods of concept generation and selection techniques were applied to this original design project. The manual concept generation activities required the team to meet outside of class and devote several hours of research and brainstorming to complete. The concepts that the team generated manually during these activities are compared to the results returned in a few minutes using each of the three automated tools in Section 6.4.3.

6.4.1. Chi-Matrix Background.

The chi-matrix method relies on a catalog of design information that stores components and the functions they perform (Strawbridge, 2002). When a designer desires to generate concepts for a given design problem, a filter matrix is used which contains only the functions needed for the given problem. This filter is multiplied into the aggregate function-component matrix to produce a matrix that contains only components that solve the needed functions. In this way a designer can generate possible solutions without having to search the entire store of knowledge manually.

6.4.2. Description of Case Study. The Rolla Area Sheltered Workshop employs persons with mental and physical disabilities to package variety boxes of dog and cat food sample packets for a local pet food manufacturer. In the interest of increased productivity and a reduced incidence of repacking, a counting and packaging assistive device was sought. The design team began by observing the previous method of packaging used by the employees and interviewing the Workshop managers to develop an understanding of the design problem and determine and weight the needs of the customer. Next, the team established the functional requirements for the design by developing a black box model and subsequent functional model, which incorporated the previously described Functional Basis terms.

A black box model is a simple representation of product’s function with input/ output flows, which are identified from the customer needs. In the model, the product is treated as a closed system and does not include the details of the flows and functions that are internal to the product; only flows input into and output from the product are taken into consideration. Figure 6.24 shows a black box model created for the sample packet counting product. After the black box model was defined, each input flow was then associated with subfunctions that operate on the flow and then aggregated to form a functional model.

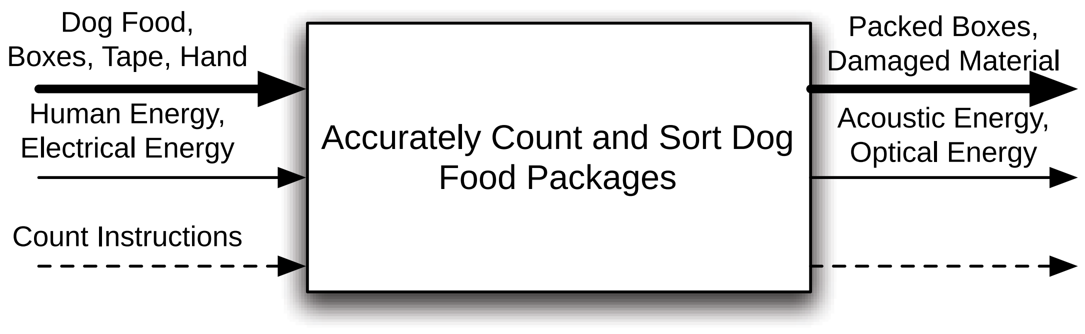


Figure 6.24. The black box model developed for the dog food packaging device.

A functional model is a description of a product or process in terms of the elementary functions that are required to achieve its overall function or purpose. A graphical form of a functional model is represented by a collection of subfunctions connected by the flows on which they operate. This structure is an easy way for a designer to see what functions must be performed without being distracted by any particular form the artifact may take. A functional model of the dog food packaging device is shown in Figure 6.25.

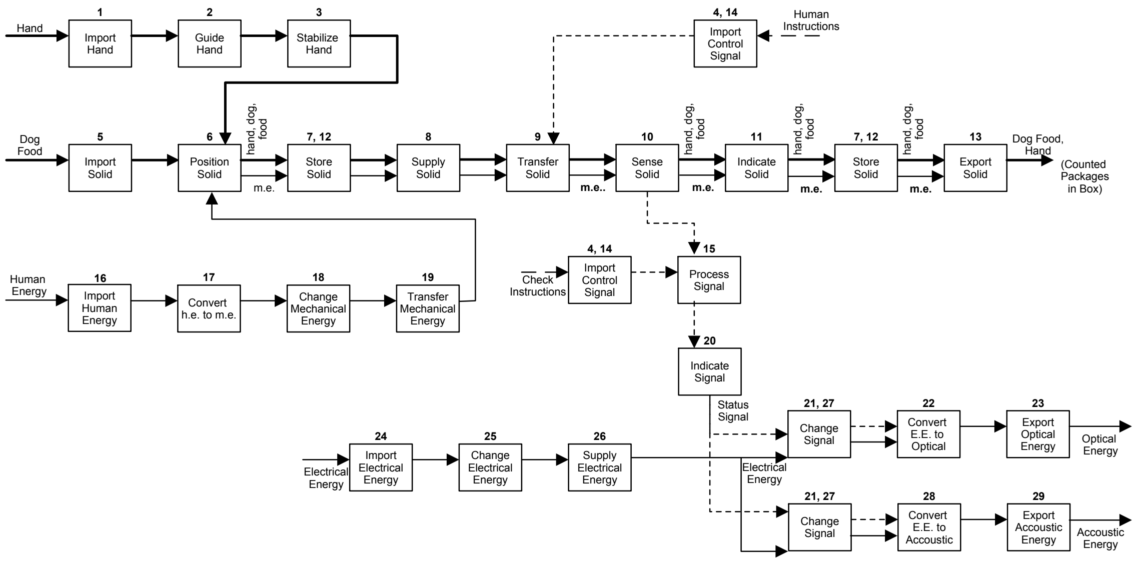
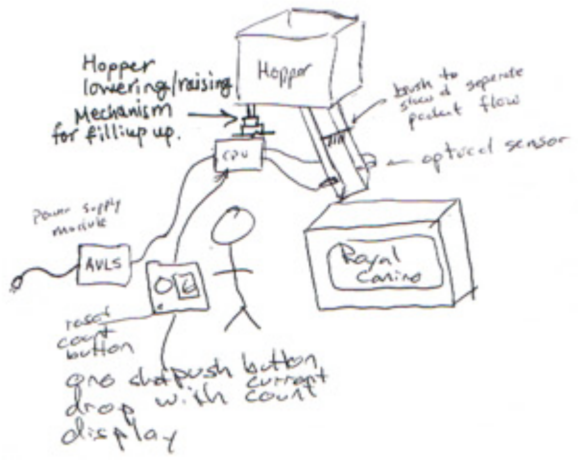
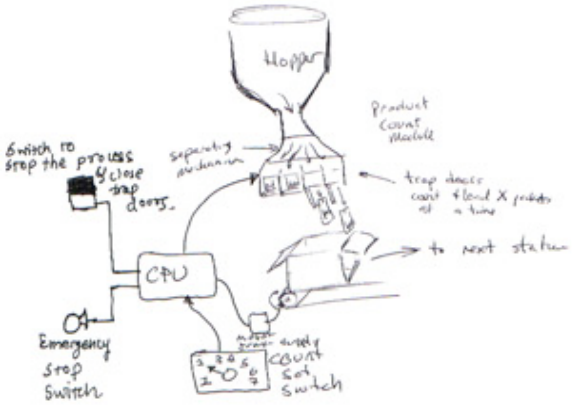


Figure 6.25. The functional model developed for the dog food packaging device.

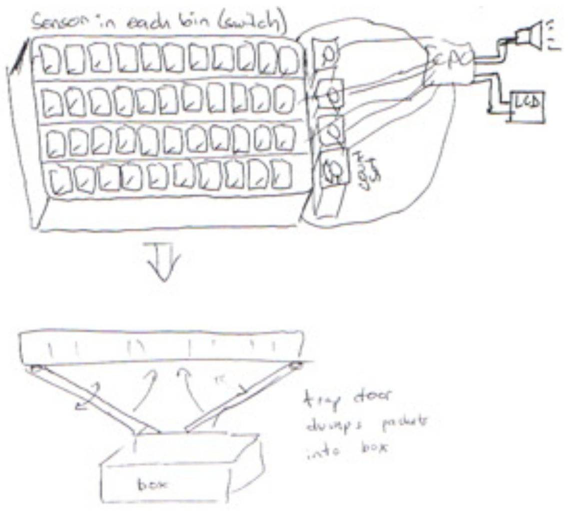
Next, the design team used many different manual concept generation techniques including the C-sketch method, Design by Analogy, the Chi Matrix approach, described above in Section 6.4.1, and the Morphological Matrix approach, described in Section 2.4.3, to explore many different creative solutions and to generate a broad spectrum of complete design concepts. The team generated five design concepts using the C-Sketch method. Three of the concepts were based on mechanical and electrical systems to transport and count the dog food packets. The fourth concept contained no moving parts or electronics and was a simple plastic tray with color-coded slots. The fifth concept built on concept four by adding switches and buzzers to indicate when the slots were full. Figure 6.26 shows three such concepts developed using the C-Sketch method (C-sketch 1, C-sketch 3 and C-sketch 5 respectively).


Hopper
Hopper lowering/raising Mechanism for fillup up.
brush to slow d seperate product flow
CPU
optical sensor
Power supply module
AVLS
Royal Canin
reset count button
one drop push button with display
current count

C-Sketch 1



C-Sketch 2

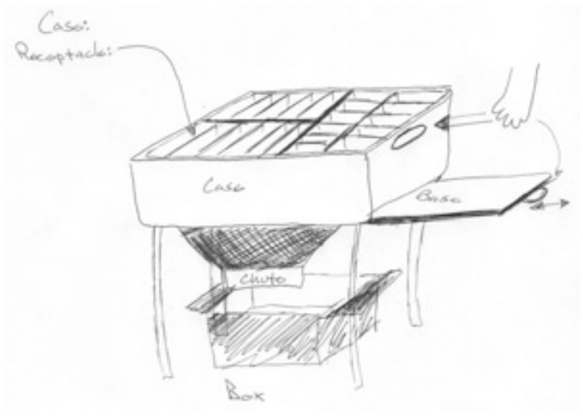

Sensor in each bin (switch)
CPU
LCD
trap door dumps products into box
box

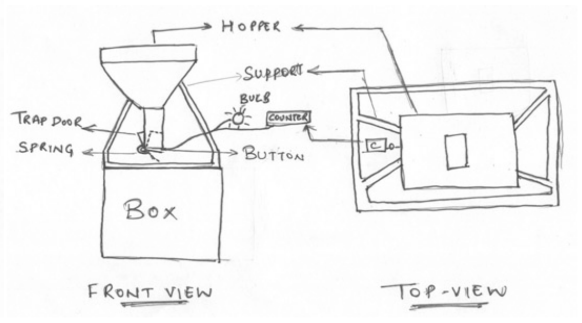
C-Sketch 3

Figure 6.26. Concepts generated by the C-Sketch method.

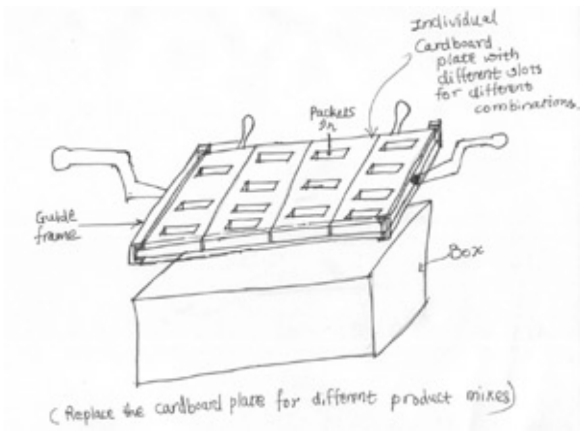
Four concepts were produced using the Design by Analogy method. The first three concepts were electro-mechanical devices using conveyors and sensors to count and transport the dog food packets. The fourth concept was a plastic tray variant with rotating handles to empty the counted dog food packets directly into the box.

Employing the Chi Matrix approach yielded five additional concepts. The first concept was based on a case with individual dog food packet receptacle slots. A sliding door was placed beneath the receptacles and was used to empty the slots once they are filled directly into the packing box via a chute. The remaining four concepts incorporated fairly simple electronics to act as counters while dog food packets were manually placed in the box. Figure 6.27 shows concepts Chi-Matrix 1, Chi-Matrix 2, and Chi-Matrix 5 as example solutions generated by the design team using this method. Although the method is similar to the Morphological Matrix Search method discussed in the remainder of this paper, the Chi Matrix solutions here were produced by hand using a different set of data.





Chi Matrix 1 Chi Matrix 2


Individual Cardboard plate with different slots for different combinations.
Packets in
Guide frame
Box
(Replace the cardboard plate for different product mixes)

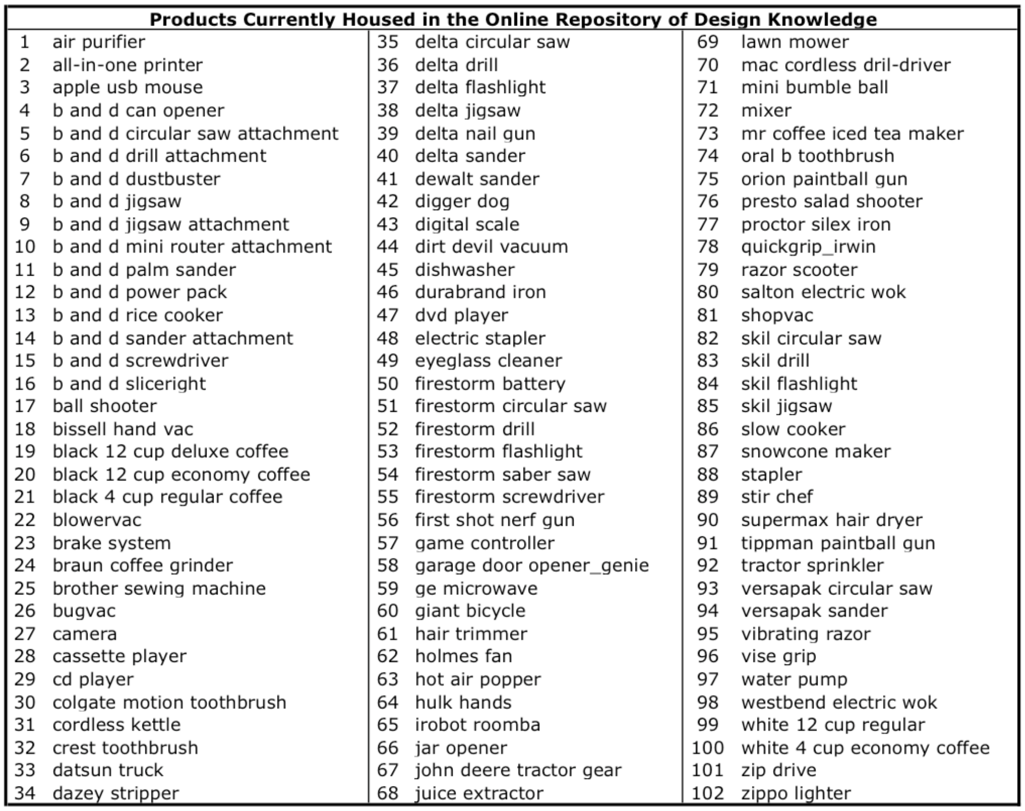
Chi Matrix 5

Figure 6.27. Concepts generated by the Chi Matrix method.

Ten concepts were generated using the morphological matrix approach. All of these concepts made use of electrical and mechanical devices to count and transport the dog food packets. It is important to recognize at this point that this morphological matrix was generated by hand by the design team and is not derived from the same data as the Morphological Matrix Search operation discussed in the rest of the paper.

6.4.3. Evaluation of the Three Automated Methods. Using the functional model shown in Figure 6.25 as input, results from each of the three automated methods, the web-based morphological matrix method described in Section 2.4.3 and the list-based and interactive morphological matrix software implementations described in Section 4, were compared to the conceptual solutions manually generated by the students for the dog food packet counter case study. The data used to generate the automated solutions was produced from the online repository of product knowledge described in Section 2.4, which currently houses detailed information on the 102 consumer products listed in Table 6.9.

Table 6.9. Information on these 102 products is currently contained within the data repository.



Products Currently Housed in the Online Repository of Design Knowledge					
1	air purifier	35	delta circular saw	69	lawn mower
2	all-in-one printer	36	delta drill	70	mac cordless dril-driver
3	apple usb mouse	37	delta flashlight	71	mini bumble ball
4	b and d can opener	38	delta jigsaw	72	mixer
5	b and d circular saw attachment	39	delta nail gun	73	mr coffee iced tea maker
6	b and d drill attachment	40	delta sander	74	oral b toothbrush
7	b and d dustbuster	41	dewalt sander	75	orion paintball gun
8	b and d jigsaw	42	digger dog	76	presto salad shooter
9	b and d jigsaw attachment	43	digital scale	77	proctor silex iron
10	b and d mini router attachment	44	dirt devil vacuum	78	quickgrip_irwin
11	b and d palm sander	45	dishwasher	79	razor scooter
12	b and d power pack	46	durabrand iron	80	salton electric wok
13	b and d rice cooker	47	dvd player	81	shopvac
14	b and d sander attachment	48	electric stapler	82	skil circular saw
15	b and d screwdriver	49	eyeglass cleaner	83	skil drill
16	b and d sliceright	50	firestorm battery	84	skil flashlight
17	ball shooter	51	firestorm circular saw	85	skil jigsaw
18	bissell hand vac	52	firestorm drill	86	slow cooker
19	black 12 cup deluxe coffee	53	firestorm flashlight	87	snowcone maker
20	black 12 cup economy coffee	54	firestorm saber saw	88	stapler
21	black 4 cup regular coffee	55	firestorm screwdriver	89	stir chef
22	blowervac	56	first shot nerf gun	90	supermax hair dryer
23	brake system	57	game controller	91	tippman paintball gun
24	braun coffee grinder	58	garage door opener_genie	92	tractor sprinkler
25	brother sewing machine	59	ge microwave	93	versapak circular saw
26	bugvac	60	giant bicycle	94	versapak sander
27	camera	61	hair trimmer	95	vibrating razor
28	cassette player	62	holmes fan	96	vise grip
29	cd player	63	hot air popper	97	water pump
30	colgate motion toothbrush	64	hulk hands	98	westbend electric wok
31	cordless kettle	65	irobot roomba	99	white 12 cup regular
32	crest toothbrush	66	jar opener	100	white 4 cup economy coffee
33	datsun truck	67	john deere tractor gear	101	zip drive
34	dazey stripper	68	juice extractor	102	zippo lighter

Ten of the 31 concepts developed during the bulk-packaging device project were chosen to compare to the morphological search results, list-based automated concept generator results, and interactive morphological matrix results. The concepts are named for the technique that was used for their generation. For example, “Chi-Matrix 1” corresponds to the first concept developed by using the Chi Matrix approach. The concepts named “Chi-Matrix 1”, “Chi-Matrix 2”, “Chi-Matrix 4”, “Chi-Matrix 5”, and “C-Sketch 5” were identified by the original design team as their top-five concepts. The remaining concepts were selected from the pool of 31 total concepts because they represented well-documented complete design solutions with definable functionality.

6.4.3.1. Survey of the Data Contained within the Repository. A function-component matrix (FCM) was downloaded from the online repository to get an initial snapshot of the coverage that the repository had in reference to the input functional model for the dog food packaging device. Of the 29 subfunctions identified for the bulk-packaging device, all 29 of the subfunctions were contained within the FCM produced from the 102 consumer products.

6.4.3.2. Preparing the Manual Concepts for Comparison. In order to compare the results from each of the automated design tools to the concepts manually developed for the bulk-packaging device, the concept sketches and design notes from the design project were revisited. Since the subfunctions used as input into each of the automated design tools originates from the initial functional model of the bulk-packaging device, each manually produced concept was checked against the same set of subfunctions. Some differences exist between the subfunctions identified in each of the concepts and those of the original functional model. This subfunction variation is partially due to the natural progression of the design process where customer needs are refined and the product direction is better identified. Table 6.10 shows a mapping of the originally identified subfunctions to each of the concepts used as a comparison in this study.

Next, each manually created concept was analyzed to determine the component that solves each subfunction found in that concept. These results were placed into a concept-specific function-component matrix for each manually generated concept to assist comparison with the results from each of the automated design tools. Table 6.11 demonstrates this idea by showing the identified subfunctions and components for the Chi-Matrix 1 concept comparison. Note that the components listed in the columns represent only those components that were identified as part of the Chi-Matrix 1 concept. Components that were identified to solve a specific function are denoted with a cell entry of 1. Shaded functions identify the subfunctions from the original functional model, which were embodied in the Chi Matrix 1 manually generated concept.

Table 6.10. Subfunctions from the original model that are embodied in each manually generated solution compared.

	Dominant Functions Identified in Manually Generated Concepts	Chi-Matrix 1	Chi-Matrix 2	Chi-Matrix 4	Chi-Matrix 5	C-Sketch 1	C-Sketch 2	C-Sketch 3	C-Sketch 4	C-Sketch 5	DBA 1
1	import human material	**1**	0	0	**1**	0	**1**	0	**1**	0	**1**
2	guide human material	**1**	0	0	**1**	0	**1**	0	0	0	**1**
3	stabilize human material	**1**	0	0	**1**	0	**1**	0	0	0	**1**
4	import control signal	0	0	0	0	0	0	**1**	**1**	**1**	0
5	import solid material	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
6	position solid material	**1**	**1**	**1**	**1**	**1**	0	**1**	**1**	**1**	0
7	store solid material	**1**	**1**	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**
8	supply solid material	**1**	**1**	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**
9	transfer solid material	**1**	**1**	0	**1**	**1**	**1**	**1**	0	0	0
10	sense solid material	0	**1**	**1**	0	**1**	**1**	**1**	0	**1**	0
11	indicate solid material	**1**	**1**	**1**	0	**1**	**1**	0	0	**1**	**1**
12	store solid material	**1**	**1**	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**
13	export solid material	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0	**1**
14	import control signal	0	0	0	0	0	0	**1**	**1**	**1**	0
15	process control signal	0	**1**	**1**	0	**1**	**1**	**1**	0	**1**	0
16	import human energy	**1**	0	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**
17	convert human energy to mechanical energy	**1**	0	0	**1**	0	0	0	0	0	**1**
18	change mechanical energy	**1**	0	0	**1**	0	0	**1**	0	0	**1**
19	transfer mechanical energy	**1**	0	0	**1**	0	0	0	0	0	0
20	indicate status signal	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
21	change status signal	0	**1**	**1**	0	**1**	**1**	**1**	0	**1**	0
22	convert electrical energy to optical energy	0	**1**	**1**	0	0	0	**1**	0	0	0
23	export optical energy	0	**1**	**1**	0	0	0	0	0	0	0
24	import electrical energy	0	**1**	**1**	0	**1**	**1**	**1**	0	**1**	**1**
25	change electrical energy	0	**1**	**1**	0	**1**	**1**	**1**	0	**1**	**1**
26	supply electrical energy	0	**1**	**1**	0	**1**	**1**	**1**	0	**1**	**1**
27	change status signal	0	**1**	**1**	0	**1**	**1**	**1**	0	**1**	0
28	convert electrical energy to acoustic energy	0	0	0	0	0	0	0	0	**1**	0
29	export acoustic energy	0	0	0	0	0	0	0	0	**1**	0

Table 6.11. Concept-specific function-component matrix for the manually generated solution labeled Chi Matrix 1.

Chi Matrix 1 Concept Specific Function-Component Matrix		Cover	Guide	Handle	Hinge	Housing	Reservoir	Support
1	import human material	0	0	**1**	0	0	0	0
2	guide human material	0	0	**1**	0	0	0	0
3	stabilize human material	0	0	**1**	0	0	0	0
4	import control signal	0	0	0	0	0	0	0
5	import solid material	0	**1**	0	0	0	0	0
6	position solid material	0	0	0	**1**	0	0	0
7	store solid material	0	0	0	0	**1**	0	0
8	supply solid material	0	**1**	0	0	0	0	0
9	transfer solid material	**1**	0	0	0	0	0	0
10	sense solid material	0	0	0	0	0	0	0
11	indicate solid material	0	0	0	**1**	0	0	0
12	store solid material	0	0	0	0	0	**1**	0
13	export solid material	0	**1**	0	0	0	0	0
14	import control signal	0	0	0	0	0	0	0
15	process control signal	0	0	0	0	0	0	0
16	import human energy	0	0	**1**	0	0	0	0
17	convert human energy to mechanical energy	**1**	0	0	0	0	0	0
18	change mechanical energy	0	0	**1**	0	0	0	0
19	transfer mechanical energy	0	0	0	0	0	0	**1**
20	indicate status signal	0	0	0	0	0	0	0
21	change status signal	0	0	0	0	0	0	0
22	convert electrical energy to optical energy	0	0	0	0	0	0	0
23	export optical energy	0	0	0	0	0	0	0
24	import electrical energy	0	0	0	0	0	0	0
25	change electrical energy	0	0	0	0	0	0	0
26	supply electrical energy	0	0	0	0	0	0	0
27	change status signal	0	0	0	0	0	0	0
28	convert electrical energy to acoustic energy	0	0	0	0	0	0	0
29	export acoustic energy	0	0	0	0	0	0	0

6.4.3.3. Comparison of Automated Results to Manually Generated Concepts.

Once the subfunction-component solution data had been extracted from each manually developed concept, the subfunctions from the functional model shown in Figure 6.25 were entered into each of the automated design tools. The returned results for the entire set of 29 subfunctions in the original model were compared to each of the concept-specific function-component matrices to determine what percentage of the manually generated concept was returned by each automated conceptual design tool.

Table 6.12 shows a summary of the comparisons between each manually generated concept and automated results for each of the three tested design tools. For the concept Chi-Matrix 1, 80.00% of the components used in the manually created concept were returned by the morphological search while only 73.33% of the components were found using the list-based concept generator and 66.67% by the interactive morphological search concept generator. This means that 80.00% of the manually derived concept (using no database of existing design knowledge) could have been derived by using the morphological search feature of the repository and 66.67%–73.33% could have been found or assembled using one of the concept generators. Analysis of all of the concepts indicate that an average of 80.44% of the ten manually derived concepts could have been automatically generated by the repository's morphological search feature, while only an average of 61.10% of the concepts could have been developed directly from the list-based automated concept generator and 53.10% directly from the interactive morph search concept generator.

Table 6.12 Portion of manually generated concept returned by each of the automated design tools.

Manual Concept	Morph Search	List-Based Concept Generator	Interactive Morph Search Concept Generator
Chi-Matrix 1	80.00%	73.33%	66.67%
Chi-Matrix 2	75.00%	43.75%	37.50%
Chi-Matrix 4	69.23%	38.46%	23.08%
Chi-Matrix 5	85.71%	71.43%	64.29%
C-Sketch 1	81.25%	62.50%	43.75%
C-Sketch 2	81.25%	56.25%	43.75%
C-Sketch 3	80.00%	60.00%	53.33%
C-Sketch 4	87.50%	87.50%	87.50%
C-Sketch 5	77.78%	44.44%	44.44%
DBA-1	86.67%	73.33%	66.67%
Average:	*80.44%*	*61.10%*	*53.10%*

Table 6.13. (Below) Summary of the number of solutions returned by each of the automated design tools.

	Morph Search	List-Based Concept Generator	Interactive Morph Search Concept Generator
Total Number of Possible Solutions	7.04E+11	6.52E+11	8.76E+09
Average Number of Solutions per Subfunction	11.21	10.93	8.45

The results returned by each method were also analyzed to determine the number of *complete* solutions possible for a user to assemble. These results are summarized in Table 6.13. The morph search returns an unfiltered matrix of component solutions yielding a total of 7.04 x 10^{11} possible solutions with an average of 11.21 possible solutions returned for each subfunction. The filtering of infeasible concepts based on historical compatibility within the concept generator reduces the number of possible complete solutions down to 8.76 x 10^{9} with and average of 8.45 solutions returned for each subfunction.

7. CONCLUSIONS

7.1. INTRODUCTION

The creative nature of design generation demands skills from a designer that must be developed and refined through practice. Advancement in technology is usually made by building on previous experiences and learning from past successes and failures. However, this knowledge transfer in the broad field of product design is difficult to accomplish. Often, few records are kept cataloging a designer’s rationale during the decision-making processes that lead to the embodiment of a successful design solution. Additionally, although many successful designs are easily identifiable, it can be unclear why or how that success materialized without prior experience dissecting or designing a similar product.

The research presented here provides a computational link between existing design tools used to gather and organize customer needs and tools used to capture and manipulate a designer’s sketches for further development using CAD software. This design tool seeks to assist a designer during the conceptual phase of the design process with computer software capable of searching a large database of design knowledge and delivering multiple relevant and easily identifiable solutions for a design problem. The search is facilitated by accepting standard input generated by a designer during a structured design process. The following sections will summarize the research presented, discuss key findings and conclusions reached during the course of the research, enumerate key contributions of the work presented here, and establish future work that will further expand the usefulness and applicability of the automated design tool presented in various design situations.

7.2. SUMMARIES AND DISCUSSIONS

The following subsections give summaries and discussions for each of the chapters contained in this dissertation.

7.2.1. Automated Concept Generation Design Tools. Section 3 and Section 4 present the algorithm and software implementation of an automated, mathematically-based concept generation technique developed from an empirical study of consumer products. Intending to facilitate the generation and evaluation of feasible concept variants during the early phases of the design process, the goal is to utilize existing design knowledge to rapidly produce a large array of concepts early in the design process. The automated concept generation method not only produces numerous results, but also has the capability to automatically rank the returned concepts based on a designer's desired specifications. One particular strength of the created algorithm is the generality it allows in terms of input and output. Unlike many other research efforts into automated concept generation which focus on the dynamic aspects of a design by utilized bond graphs (e.g. Welch and Dixon, 1991; Gui and Mäntylä, 1994; Bradley, *et al.*, 1993; Oh, *et al.*, 1996; Bracewell and Sharpe, 1996; Sieger and Salmi, 1997) or focus on applicability to specific design situations (e.g. Yates and Beaman, 1995; Hayes, 1995; Finkelstein, 1998), the design tool presented here allows for varying types of design input to be entered and varying categories of design solutions to be combined into full solutions. In addition, compared to traditional concept generation methods, the process presented here is quick and does not require the effort of an entire team of designers. Furthermore, ongoing research activities seem to indicate that the concept generation software is capable of producing concept variants comparable to those produced manually by upper-class undergraduate engineering students.

7.2.2. Component Classification Hierarchy and Procedure. The research described in Section 5 outlines a hierarchical framework constructed to help guide the classification of components and extend previously presented work toward a component naming convention that led to a flat list of 114 distinct generic component terms (Kurtoglu, *et al.*, 2005). In addition, the framework presented uses primary and secondary levels of specification coupled with a robustly defined procedure to help identify the appropriate placement of terms into the hierarchy while maintaining the goals of completeness and exclusivity in component coverage. Under this proposed framework, components of widely varying levels of complexity (e.g. an electric wire vs. an electric motor) may both be placed within the hierarchical structure, as long as the black box functionality may be limited to a single function contained within the Functional Basis list of terms. Additionally, components that exhibit functionality directly vital to the functioning of a product (e.g. a plug and cord) are not distinguished from components that only exhibit functionality that supports the function of a product in a more indirect manner (e.g. a bracket that secures an electric motor in place). Finally, although component definitions include references to component form as a way to distinguish between the various component “species”, information regarding a component’s form or method of manufacture is not used within the component hierarchy. For the components classified thus far, complexity, type of functionality (i.e. whether it directly or indirectly works to solve conceptual functionality), and other characteristics not function related do not seem to negatively impact the effectiveness of the proposed framework. However, as the number of component "species" grows, the proposed framework could be easily adjusted to fit into a larger hierarchical framework where other component characteristics that are deemed appropriate may be added as super-groups to the proposed hierarchy (see Figure 7.1). As with the classification of living organisms, the classification of components is an endeavor that will be strengthened by discourse.

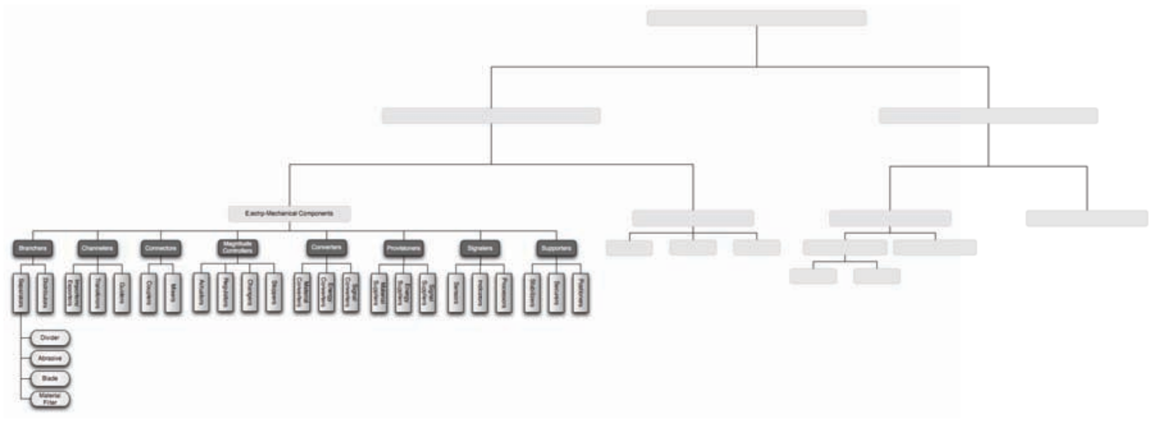


Figure 7.1. The proposed hierarchy has the potential to be adapted to a larger structure if components from other domains do not fit within the structure proposed for electromechanical devices from consumer products.

In addition to establishing a method of consistently achieving complete and exclusive coverage of the component space, the hierarchical ontology also establishes a means to distinguish traditionally similarly named components that, in fact, have very different functionality. Just as a black-tailed prairie dog (which is, indeed, not a dog at all) and a common domesticated dog could be distinguished as unrelated by their scientific names (i.e. *Cynomys ludovicianus* and *Canis lupus familiaris*), a similar formal naming structure could be used to distinguish common component names that may be misleadingly similar (e.g. a wheel used as a control device to steer a car vs. a wheel that is fixed to an axle and allows for an object, such as a bicycle, to roll along the ground). As with animal naming, the formal names may be used when clarity of meaning is essential, while the familiar names would not lose their meanings.

Since the primary motivation behind the creation of an effective component ontology is to assist designers during the early phases of design, a hierarchy organized by functional purpose incorporates a level of abstraction that will allow functionally similar but distinct components to be considered for a design. By following the presented procedure and utilizing the proposed hierarchical structure where components are grouped together by functional purpose and distinguished by form and functional embodiment, it is postulated that the goals of completeness and exclusivity of term coverage will also be effectively maintained.

7.2.3. Experimental Activities and Case Studies. Section 6 presents several research activities designed to test the effectiveness of the proposed automated concept generation tool throughout various stages of its development. First, research activities performed by four undergraduate researchers at the University of Texas at Austin and the University of Missouri–Rolla to evaluate the early list-based form of the concept generator, described in Section 4.2, is presented. Included are a qualitative investigation performed by a group of four undergraduate researchers at the University of Texas at Austin and the University of Missouri-Rolla and a post-investigation quantitative analysis designed to evaluate the list-based concept generator. Overall, the analyses described demonstrate that even this early version of the implemented concept generator algorithm holds promise as a useful design tool. The investigations presented in this section identified many paths for further development of both the software implementation as well as the design tools used to support this automated method of concept generation, including the design repository and Component Basis. One identified avenue of development for the early concept generator software, later incorporated into the second generation implementation, was enabling a user to submit a full functional model (with branching chains and multiple input and output flows). Another potentially useful user-interface improvement, later implemented, was to output the generated design solutions as a more interactive tool instead of listing the results in a ranked list of solution chains. The interactive morphological matrix style output, described in Section 4.4, allows a designer to “tinker” with solution variations rather than be presented with an overwhelming list of solutions that may contain groups of variants with only mild deviations from each other. Thus a designer is free to choose various configurations and get instant feedback on compatibility and ranking scores on a selected design, since metrics such as measures of failure, manufacturing and assembly costs, quality, recyclability, or some mathematical combination of similar design characteristics can be embedded in the seed FCM and DSM that seed the concept generator. In general, management of the design solutions, including developing useful ranking schemes and grouping similar solutions into sets, will be a key area of development, since this aspect of the software strongly influences a designer's perception of the software's usefulness.

Section 6.3 presents a qualitative investigation performed by a group of three designers at the University of Texas at Austin and the University of Missouri-Rolla and a post-investigation quantitative analysis designed to evaluate multiple parameters within a systematic design process; functional requirement abstraction level and manual versus automated concept generation in original and redesign scenarios. The analyses described in this section demonstrate that, as with any tool, a computerized design tool must either be intuitive enough to use that a designer can naturally incorporate it into the design process, or the benefits of using the software must be great enough to justify scaling a learning curve to reap the advantage. As Snowden (Andrews and Snowden, 2002) states, "...technology [is] a tool: If you pick it up and it fits in the hand, then it's useful. If you have to bio-reengineer your hand to fit your tool, it's a waste of time." To this end, as with all useful computer applications, the automated tools proposed must be refined so that, from a designer's perspective, the tool does not hinder the design process. The research presented was performed to help compare the current effectiveness of the automated design tool and guide the further development of the method into a useful computational conceptual design tool.

The case study described in Section 6.4 investigates the results returned by the existing web-based morphological search tool described in Section 2.4.3, the list-based concept generation implementation described in Section 4.2, and the interactive concept generation software described in Section 4.4. Each of the three design tools were evaluated against manually created concepts (using no database of existing design knowledge) generated for a dog food packaging device. The web-based morphological search tool captured an average of 80.44% of the ideas manually generated, while each of the automated concept generator tools captured between 53.10% and 61.10% of the manually generated ideas after filtering the incompatible solutions from those returned by the morphological search method.

A key characteristic of the two automated concept generators compared during this study is the filtering of *incomplete* solutions from the pool of concepts automatically produced based on data contained within the database of design knowledge. Although the filtering out of incomplete solutions begins to dramatically reduce the pool of automatically generated solutions that a designer must parse through (in this study a reduction from 7.04×10^{11} to 8.76×10^{9} possible solutions), many feasible partial solutions are lost, as indicated by the reduced hit percentage between the web-based morphological search and the two automated concept generators. However, the increased number of “misses” by each of the concept generator design tools can be addressed by refining several existing traits of the data contained in the web-based design repository.

First, the data contained in the repository may include intermediate component connections to link together the major components identified in the manually generated solutions. For instance, solution pairs comprised of a battery connected to a circuit board were filtered out because none of the products that are dissected and stored within the repository have a battery directly in physical contact with a circuit board. This fact alone accounts for significantly decreased correlation between the automated solutions produced by the concept generators and the manually generated solutions labeled Chi Matrix 2, Chi Matrix 4, C-Sketch 1, C-Sketch 3, and C-Sketch 5. This problem could be alleviated by using a secondary method of compatibility identification beyond direct physical contact, e.g. by identifying compatible input and output flow ports for a component.

Other problems arise because of “bottlenecking” of solutions for a particular subfunction. That is to say, if the repository data for a given subfunction is limited, the results returned by any of the methods may produce only a single or very few solutions. This has a dramatic effect on the morphological search automated concept generator especially, because all solutions returned should be complete and therefore must include the same solution for the restricted function. The best way to avoid this problem is to continue to populate the design repository with many products from a variety of domains and complexities.

The final problem identified as having a significant impact on the reduced hit return from the automated concept generators relates directly to the signal flows contained within the input functional model. Product knowledge entered into the online repository for components that primarily have functionality dealing with signal flows through a product suffers from inconsistencies that are not as readily seen when dealing with components that mainly deal with materials and energies moving through a product. Recent research at the University of Missouri–Rolla has made strides to develop grammars to address the significant issues of inconsistency in modeling signal flows, but this research has not yet been retroactively applied to products already contained in the repository.

The design tools investigated in this study offer designers an additional approach for generating concept variants and presents historically recorded subfunction solutions. The high hit percentages for the morphological search further reinforces this feature of the online design repository as a promising tool for concept generation. The lower hit percentages for the two automated concept generators (the list-based version–Section 4.2., and interactive version–Section 4.4.) that limit the results returned to only feasible solutions based on component compatibility suggest that a larger pool of data is needed in order to avoid limiting the results with obstacles such as solution “bottlenecking” and data inconsistency. However, it is important to note that although these obstacles did have an impact on the commonality percentages calculated for the interactive morphological search, many complete and physically feasible solutions were returned by the automated tools that the students did not manually generate.

Two distinct advantages emerge from the use of the automated design tools. First, the process is automated to the extent that component solutions are identified computationally through repeatable algorithms rather than through mental retrieval. Secondly, the aggregation of knowledge represented in a generated matrix offers a greater degree of diversity, permanence, and portability than human recollection alone is likely to provide. The process for retrieving knowledge from each of the design tools is quick and does not require the efforts of an entire design team.

7.3. KEY CONTRIBUTIONS

The computational theory of concept generation addresses the lack of automated methods in the early stage of conceptual design. It is based on the notion that archived product design knowledge can be reused to create new product concepts. The theory behind the design knowledge relationships is sound, drawn from accepted systematic design methodologies. The representation of the various design knowledge relationships in a mathematical form is a rather novel development. The formulation of a theoretical construct to compute concept variants from archived knowledge breaks new ground by helping to push engineering design concept generation activities into the realm of artificial intelligence.

One of the key advantages of the computational theory of concept generation is that it sets forth a path to capture and reuse corporate knowledge. This is a particular useful notion for industry where design knowledge often resides in the minds of the more experienced designers. This approach provides a way to capture abstract and specific product design knowledge (in the form of a design repository) and transfer that knowledge to less experienced designers (through browsing the repository and computing new concept variants from the concept generator). Likewise, design education can benefit from this approach in the education of engineering designers.

Another key advantage of this approach is that the supporting knowledge base can grow and adapt over time. As more and more product knowledge is accumulated in a repository, the greater the breadth (or depth, for that matter) of potential concept variants becomes. A question that results from this is how much data is necessary to make this concept generation algorithm pliable? Preliminary tests within our lab show that the design knowledge in the UMR repository can generate 60-80% of the concept variants that human design teams produce (the comparisons were made after the human designers completed their design projects). In addition to the overlapping concept variants, the concept generator algorithm produces two to four times more concept variants that are viable than design teams. These results are based on a repository knowledge base of 102 products.

There can be too much of a good thing, however. The output of the concept generator algorithm can reach into the tens of thousands of viable concept variants, depending on the size of the input functional models and the make up of the knowledge base. Ranking quickly becomes a critical method to further filter the viable concept variants into a more manageable set. Any number of ranking approaches is possible with the types of design knowledge stored in the repository, as the concept generator approach does not preclude or dictate any particular type.

7.4. FUTURE WORK

The following subsections outline future research projects that could further enhance the proposed computational design tool and its supporting technologies.

7.4.1. Extensions to the Automated Concept Generator. Although the research presented in this dissertation has demonstrated usefulness for early design concept generation, the effectiveness of the tool would benefit from additional research. For instance, since conceptual design is inherently an evolutionary process, significant benefits could be gained by further extending the dynamic functionality of the software. When a design solution is first explored, a core set of desired functionality is known by the designer, but the very act of choosing a specific solution component begins to add additional functional requirements to a design. In its current form, the proposed computational theory allows for an initial static set of requirements to be input, but does not support these evolutionary changes that all designs undergo.

Additional benefits would be gained by further enhancing the presentation of solutions to a designer. The interactive morphological search takes strides in the direction of giving a design real-time feedback on the compatibility of a solutions, but focusing research on taking the text-based solutions and presenting them in a visual manner (i.e. creating a virtual prototype) would make a significant impact on how a designer interacts with the knowledge presented. The computational theory presented in this dissertation also very readily could be extended to present a designer with design modules by employing the method of clustering components into design groups presented by Kusiak and Szczerbicki (1993).

7.4.2. Ranking and Identifying "Good" Designs. Features of future software versions should include the exploration of various ranking methods to help sort the concept variants generated. Although using the design structure matrix as a first-pass filter eliminates many less useful concepts from the set of design variants, metrics such as measures of failure, manufacturing and assembly costs, quality, recyclability, or some mathematical combination of similar design characteristics could prove to be valuable tools for identifying the most promising variants among the hundreds (or thousands) of potentially viable solutions found. In general, management of the design solutions, including developing useful ranking schemes and grouping similar solutions into sets, will be a key area of development, since this aspect of the software will strongly dictate whether the introduced design tool will help or hinder the design process from the designer's perspective.

7.4.3. Early Design Tools for Multiple Design Contexts. The computational theories presented in this dissertation, because they are generalized, have the potential to impact other areas of design beyond the scope of product design. Investigations into the different approaches and requirements that designers in different contextual situation face may demonstrate that the established theory is adaptable to situations such as designing process layouts (e.g. for manufacturing purposes or potentially even chemical or biological (protein) design processes), dictating performance parameters for complex systems (i.e. integration designs), automating previously manual processes, designing efficient workflow layouts.

7.4.4. Component Classification Research. Further areas of improvement for the established component templates and classification procedure includes establishing more complete port templates that may be used to help build up more complete conceptual ideas during the early stages of conceptual design. By knowing the number and types of ports a component term typically has, software may be used to help guide the evolution of a full conceptual idea, including parts needed to indirectly support the functionality of other components. Additionally, design measure estimates (such as measures of potential failures, manufacturability, cost, size, performance, etc.) could be determined across each component group and used to help guide concept selection early in the design process. Other work could include creating a forum for the discussion of new and existing component terms, their placement within the hierarchical ontology, and even the organization of the hierarchical ontology as well. Finally, the work presented here is focused mainly on components found in consumer products. Additional work should look at other design domains and identify how the hierarchy should be altered or expanded to include a broader range of component types. As with the animal groupings, the process to create a complete and robust hierarchy should be an evolutionary process with much discussion involved.

7.4.5. Other Related Research Areas. Further areas of refinement include enhancing the robustness of the data entry procedure for populating the design repository. Since the validity of the results returned by the concept generator is closely tied to the validity of the knowledge stored in the repository, the quality of returned results is sensitive to the quality and correctness of design knowledge contained in the repository. For instance, during the quantitative study of the data from the methodological comparison reported in Section 6.2, an error in data entered into the design repository was identified when the design solution of an “indicator light” turned up as a solution to the subfunction “convert electrical energy to mechanical energy.” The entry error was identified as an incorrectly selected component classification term, but since this component was also compatible with the surrounding components via the identified component connections, it was not filtered out of the compatible solutions returned by the concept generator.

7.5. PARTING WORDS.

As mentioned in Section 1, Yang (2003) concludes that it is both important to generate and solidify a large number of ideas as well as begin prototyping a design early in the design process. Anderson’s (1981) research indicates that while experienced designers tend to approach a design problem broadly at first, inexperienced designers explore solutions using a depth-first approach. From this perspective, the presented concept generation theory encourages novice designers to investigate a broad range of solutions, as a more experienced designer may be inclined to do. The matrix-based concept generator allows for the quick development of conceptual ideas and for significantly different concepts to be explored through sketching, since it utilizes the component classification scheme rather than specific component instantiations to return results. In addition, the wide array of results returned by the concept generator supports creativity and design research, which indicates that conceptual design activities should contain both divergent and convergent steps (Cross, 1994; Pugh, 1991; Guilford, 1959; Roozenburg and Eekels, 1995). The computational theory presented in this dissertation demonstrates the potential for automated technologies to support designers during the early stages of design and reuse existing design knowledge in a way that contributes to innovation and creativity in product design.

APPENDIX A

MEMIC Software Code

--start ‘ConGen.java’ code -------------------------------------

```
/*--------------------------------------------------------------------------------------------------------------
 * Concept Generator (a.k.a Memic - Morphological Evaluation Machine and Interactive Conceptualizer)
 * This software accepts the functional description of a product to be designed and outputs a morphological matrix
 * that a designer may interact with to build and evaluate conceptual solutions.
 *
 * Version: 2.0
 * Author: Cari R. Bryant, University of Missouri-Rolla
 * Last Update: July 01, 2007
 * Disclaimer: This program is used primarily as a proof-of-concept and is not developed using rigorous Java
 * development conventions.
 *------------------------------------------------------------------------------------------------------------*/
package edu.umr.ide;

import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;

import javax.swing.*;
import javax.swing.border.*;

public class ConGenV2_0 {
          /*---------------------------------------------------------------------------------------------
          * Define global variables for user GUI frames
          *---------------------------------------------------------------------------------------------*/
          JFrame baseFrame; // Frame for user input GUI
          JPanel background; // Panel for input background
          JButton loadFM, loadFCM, loadDSM; // Buttons to load Functional Model, Function-Component Matrix, and Design
                                              Structure Matrix files
          JTextField labelFM, labelFCM, labelDSM; // Labels for file and directory displays

          JFrame resultFrame; // Frame for results output GUI
          JPanel resultBackground; // Panel for output background
          ArrayList<JTextField> selectedFields = new ArrayList<JTextField>(); // Array of selected component fields
          ArrayList<JButton> removeButtons = new ArrayList<JButton>(); // Array of removeButtons
          ArrayList<ArrayList<JButton>> fullCompArray = new ArrayList<ArrayList<JButton>>(); // Array of morph matrix
                                                                                               rows

          Color lightblue = new Color(200,205,225); // Background color
          Color darkblue = new Color(142,148,191); // Dark accent color
          Color white = new Color(255,255,255); // Light accent color

          /*---------------------------------------------------------------------------------------------
          * Define global variables for program methods
          *---------------------------------------------------------------------------------------------*/
          String lastFMOpenPath = System.getProperty("user.home"); // Keeps track of path of last FM file opened
          String lastOpenPath = System.getProperty("user.home"); // Keeps track of path of last FCM or DSM file opened
          String lastSavePath = System.getProperty("user.home"); // Keeps track of path of last file saved

          ArrayList fmArray = null; // Array to hold raw FM data file data
          ArrayList fcmArray = null; // Array to hold raw FCM data file data
          ArrayList dsmArray = null; // Array to hold raw DSM data file data
          ArrayList<Subfunction> fmLinks = new ArrayList<Subfunction>(); // Array to hold link info from the functional
                                                                            model

          int fmHeader = 0; // Number of header rows before column labels in FM data file
          int fcmHeader = 0; // Number of header rows before column labels in FCM data file
          int dsmHeader = 0; // Number of header rows before column labels in DSM data file

          ArrayList<String> masterComponentList = new ArrayList<String>(); // List of all components in DSM
          ArrayList<Integer> functionsWithoutSolutions = new ArrayList<Integer>(); // List of all subfunctions with
                                                                                     unknown compatible solutions

          /*---------------------------------------------------------------------------------------------
          * Initiate program execution
          *---------------------------------------------------------------------------------------------*/
          public static void main(String[] args) {
                    ConGenV2_0 gui = new ConGenV2_0(); // Create new object
                    gui.buildInputGUI(); // Run method to build the input GUI
          } // end main()
```

```
/*------------------------------------------------------------------------------------------
 * Builds the graphical user interface for the user input into the concept generation program
 *------------------------------------------------------------------------------------------*/
private void buildInputGUI()  {
          /* Define and Initialize variables ----------------------------------------------------------*/
          baseFrame = new JFrame();
          background = new JPanel();
          loadFM = new JButton();
          loadFCM = new JButton();
          loadDSM = new JButton();
          labelFM = new JTextField();
          labelFCM = new JTextField();
          labelDSM = new JTextField();

          JPanel step1Panel = new JPanel(), step2Panel = new JPanel(), step3Panel = new JPanel();
          JPanel radioPanel = new JPanel();
          JRadioButton defaultComponents = new JRadioButton(), customComponents = new JRadioButton();
          JPanel customComponentLoadPanel = new JPanel(), loadFCMPanel = new JPanel();
          JPanel loadDSMPanel = new JPanel();
          JTextPane spacerPane = new JTextPane();
          JButton goButton = new JButton();

          /* baseFrame ------------------------------------------------------------------------------*/
          {
                    baseFrame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
                    baseFrame.setResizable(false);
                    baseFrame.setTitle("Memic (The Concept Generator v2.0)");
                    baseFrame.setBackground(lightblue);
                    Container baseFrameContentPane = baseFrame.getContentPane();
                    baseFrameContentPane.setLayout(new BoxLayout(baseFrameContentPane, BoxLayout.Y_AXIS));
                    /* background ------------------------------------------------------------------*/
                    {
                              background.setBorder(new EmptyBorder(10, 10, 10, 10));
                              background.setFocusable(false);
                              background.setMaximumSize(new Dimension(800, 320));
                              background.setOpaque(false);
                              background.setPreferredSize(new Dimension(800, 320));
                              background.setMinimumSize(new Dimension(800, 320));
                              background.setBackground(lightblue);
                              background.setLayout(new BoxLayout(background, BoxLayout.Y_AXIS));

                              /* step1Panel ---------------------------------------------------------------*/
                              {
                                        step1Panel.setBorder(new CompoundBorder(
                                                  new TitledBorder("Step 1: Import functional model."),
                                                  new EmptyBorder(5, 5, 5, 5)));
                                        step1Panel.setMinimumSize(new Dimension(800, 67));
                                        step1Panel.setFocusable(false);
                                        step1Panel.setOpaque(false);
                                        step1Panel.setBackground(lightblue);
                                        step1Panel.setLayout(new BoxLayout(step1Panel, BoxLayout.X_AXIS));

                                        /* load FM -------------------------------------------------------*/
                                        loadFM.setText("Find File");
                                        loadFM.setOpaque(false);
                                        loadFM.addActionListener(new loadDataListener());
                                        step1Panel.add(loadFM);

                                        /* label FM ------------------------------------------------------*/
                                        labelFM.setText("../");
                                        labelFM.setEditable(false);
                                        labelFM.setFocusable(false);
                                        labelFM.setBackground(white);
                                        step1Panel.add(labelFM);
                              }
                              background.add(step1Panel);

                              /* step2Panel ---------------------------------------------------------------*/
                              {
                                        step2Panel.setBorder(new CompoundBorder(
        new TitledBorder("Step 2: Load component data."),
        new EmptyBorder(5, 5, 5, 5)));
step2Panel.setFocusable(false);
step2Panel.setOpaque(false);
step2Panel.setBackground(lightblue);
step2Panel.setLayout(new BorderLayout());

/* radioPanel ----------------------------------------------------*/
{
        radioPanel.setBackground(lightblue);
        radioPanel.setLayout(new BoxLayout(radioPanel,
                BoxLayout.Y_AXIS));

        /* defaultComponents ------------------------------------*/
        defaultComponents.setText("Use default component data.");
        defaultComponents.setContentAreaFilled(false);
        defaultComponents.setBackground(lightblue);
        defaultComponents.setEnabled(true);
        defaultComponents.addActionListener(new
                defaultFilesListener());
        radioPanel.add(defaultComponents);

        /* customComponents -------------------------------------*/
        customComponents.setText("Load custom component files.");
        customComponents.setOpaque(false);
        customComponents.setContentAreaFilled(false);
        customComponents.setBackground(lightblue);
        customComponents.setSelected(true);
        customComponents.addActionListener(new
                customFilesListener());
        radioPanel.add(customComponents);
}
step2Panel.add(radioPanel, BorderLayout.NORTH);

/* customComponentLoadPanel --------------------------------------*/
{
        customComponentLoadPanel.setBorder(null);
        customComponentLoadPanel.setBackground(lightblue);
        customComponentLoadPanel.setLayout(new
                BoxLayout(customComponentLoadPanel,
                BoxLayout.Y_AXIS));

        /* loadFCMPanel -----------------------------------------*/
        {
                loadFCMPanel.setBackground(lightblue);
                loadFCMPanel.setLayout(new
                        BoxLayout(loadFCMPanel,
                        BoxLayout.X_AXIS));

                /* loadFCM ------------------------------------*/
                loadFCM.setText("Load FCM");
                loadFCM.setEnabled(true);
                loadFCM.setOpaque(false);
                loadFCM.setContentAreaFilled(false);
                loadFCMPanel.add(loadFCM);
                loadFCM.addActionListener(new
                        loadDataListener());

                /* labelFCM -----------------------------------*/
                labelFCM.setText("../");
                labelFCM.setEnabled(true);
                labelFCM.setCursor(Cursor.getPredefinedCursor
                        (Cursor.DEFAULT_CURSOR));
                labelFCM.setEditable(false);
                labelFCM.setFocusable(false);
                labelFCM.setBackground(white);
                loadFCMPanel.add(labelFCM);
        }
        customComponentLoadPanel.add(loadFCMPanel);

        /* loadDSMPanel -----------------------------------------*/
        {
```

```
                    loadDSMPanel.setBackground(lightblue);

                    loadDSMPanel.setLayout(new BoxLayout
                        (loadDSMPanel, BoxLayout.X_AXIS));

                    /* loadDSM ------------------------------------*/
                    loadDSM.setText("Load DSM");
                    loadDSM.setEnabled(true);
                    loadDSM.setOpaque(false);
                    loadDSM.setContentAreaFilled(false);
                    loadDSM.addActionListener(new
                        loadDataListener());
                    loadDSMPanel.add(loadDSM);

                    /* labelDSM -----------------------------------*/
                    labelDSM.setText("../");
                    labelDSM.setEnabled(true);
                    labelDSM.setCursor(Cursor.getPredefinedCursor
                        (Cursor.DEFAULT_CURSOR));
                    labelDSM.setEditable(false);
                    labelDSM.setFocusable(false);
                    labelDSM.setBackground(Color.white);
                    loadDSMPanel.add(labelDSM);
                }
                customComponentLoadPanel.add(loadDSMPanel);
            }
            step2Panel.add(customComponentLoadPanel, BorderLayout.CENTER);

            /* spacerPane ----------------------------------------------------*/
            spacerPane.setPreferredSize(new Dimension(50, 16));
            spacerPane.setBackground(lightblue);
            spacerPane.setAutoscrolls(false);
            spacerPane.setDragEnabled(false);
            spacerPane.setEditable(false);
            spacerPane.setEnabled(false);
            spacerPane.setFocusable(false);
            spacerPane.setOpaque(false);
            step2Panel.add(spacerPane, BorderLayout.WEST);
        }
        background.add(step2Panel);

        /* step3Panel ---------------------------------------------------------------*/
        {
            step3Panel.setBorder(new CompoundBorder(
                new TitledBorder("Step 3: Generate interactive
                    morphological matrix"),
                new EmptyBorder(5, 5, 5, 5)));
            step3Panel.setFocusable(false);
            step3Panel.setOpaque(false);
            step3Panel.setBackground(new Color(200, 205, 225));
            step3Panel.setLayout(new BorderLayout());

            /* goButton ------------------------------------------------------*/
            goButton.setText("Create concepts!");
            goButton.setContentAreaFilled(false);
            goButton.setPreferredSize(new Dimension(135, 15));
            goButton.setMinimumSize(new Dimension(135, 15));
            goButton.setMaximumSize(new Dimension(135, 15));
            goButton.addActionListener(new goButtonListener());
            step3Panel.add(goButton, BorderLayout.WEST);
        }
        background.add(step3Panel);
    }
    baseFrameContentPane.add(background);
    baseFrame.pack();
    baseFrame.setLocationRelativeTo(baseFrame.getOwner());
    baseFrame.setVisible(true);
}

/* componentRadioButtons ------------------------------------------------------------------*/
ButtonGroup componentRadioButtons = new ButtonGroup();
componentRadioButtons.add(defaultComponents);
```

```
          componentRadioButtons.add(customComponents);
} // end buildInputGUI()

/*-----------------------------------------------------------------------------------------------
* Listener to register the defaultComponents radio button
* actionPerformed method:
*         1. Disables loadFCM and loadDSM
*         2. Automatically loads data from included files into the FCM and DSM arrays
*-----------------------------------------------------------------------------------------------*/
private class defaultFilesListener implements ActionListener {
          public void actionPerformed(ActionEvent custom) {
                    // Disable manual loading buttons --------------------------------------------------
                    loadFCM.setEnabled(false);
                    loadDSM.setEnabled(false);
                    labelFCM.setEnabled(false);
                    labelDSM.setEnabled(false);

                    // Automatically load data from included files
                    ArrayList<Object> fcmFileArray = new ArrayList<Object>(); // Holds array of strings read
                                                                                 from data file
                    ArrayList<Object> dsmFileArray = new ArrayList<Object>(); // Holds array of strings read
                                                                                 from data file
                    File newFCMFile = new File("FCM.txt"); // Holds pathname for FCM data file to be read
                    File newDSMFile = new File("DSM.txt"); // Holds pathname for DSM data file to be read

                    try { // Try to read data from file
                              BufferedReader reader = new BufferedReader(new FileReader(newFCMFile));
                                        // Read stream
                              String line = null; // Initialize variable to get data
                              while ((line = reader.readLine()) != null) { // Read in data until end of file
                                        String[] splitLine = line.split("\t"); // Split line string at tabs
                                        fcmFileArray.add(splitLine); // Add split line strings to file array
                              }
                              reader.close(); // Close read stream
                    } catch (Exception e) {
                              JOptionPane.showMessageDialog(baseFrame, "Error reading file.");
                                        // Error dialog box
                              e.printStackTrace();
                    } // end try

                    try { // Try to read data from file
                              BufferedReader reader = new BufferedReader(new FileReader(newDSMFile));
                                        // Read stream
                              String line = null; // Initialize variable to get data
                              while ((line = reader.readLine()) != null) { // Read in data until end of file
                                        String[] splitLine = line.split("\t"); // Split line string at tabs
                                        dsmFileArray.add(splitLine); // Add split line strings to file array
                              }
                              reader.close(); // Close read stream
                    } catch (Exception e) {
                              JOptionPane.showMessageDialog(baseFrame, "Error reading file.");
                                        // Error dialog box
                              e.printStackTrace();
                    } // end try

                    fcmArray = fcmFileArray; // Save data to FCM global variable
                    dsmArray = dsmFileArray; // Save data to DSM global variable

          } // end actionPerformed
} // end defaultFilesListener

/*-----------------------------------------------------------------------------------------------
* Listener to register the customComponents radio button
* actionPerformed method:
*         1. Enables loadFCM and loadDSM
*-----------------------------------------------------------------------------------------------*/
private class customFilesListener implements ActionListener {
          public void actionPerformed(ActionEvent custom) {
                    loadFCM.setEnabled(true);
                    loadDSM.setEnabled(true);
                    labelFCM.setEnabled(true);
                    labelDSM.setEnabled(true);
```

```
		}
	} // end customFilesListener

	/*-----------------------------------------------------------------------------------------------
	* Listener to register any of the data file load button presses
	* actionPerformed method:
	*		1. Triggers box to select data file
	*		2. Updates file and directory label for file chosen by user
	*-----------------------------------------------------------------------------------------------*/
	private class loadDataListener implements ActionListener {
		public void actionPerformed(ActionEvent load) {
			Object source = load.getSource(); // Determines source button that triggered the listener
			ArrayList returnedArray = loadFile(); // Holds array loaded from data file

			if (returnedArray != null) { // If returned file array contains data
				File filePath = (File) returnedArray.get(0); // Get path of opened file
				returnedArray.remove(0); // Remove path from file data array
				if (source == loadFCM) { // If "Load FCM" button was pressed
					labelFCM.setText(filePath.getPath()); // Update FCM label field
					fcmArray = returnedArray; // Data from the opened file was for the
					                             FCM
				} else if (source == loadDSM) { // If "Load DSM" button was pressed
					labelDSM.setText(filePath.getPath()); // Update DSM label field
					dsmArray = returnedArray; // Data from the opened file was for the
					                             DSM
				} else if (source == loadFM) { // If "Find File" button was pressed
					labelFM.setText(filePath.getPath()); // Update FM label field
					fmArray = returnedArray; // Data from the opened file was for the FM
				}
			} // end if
		} // end actionPerformed
	} // end loadDataListener

	/*-----------------------------------------------------------------------------------------------
	* Prompts user for file location, reads in tab-delimited file
	* data, and saves the data to a matrix
	*-----------------------------------------------------------------------------------------------*/
	private ArrayList loadFile() {
		File newFile = new File(lastOpenPath); // Holds pathname for data file to be read
		JFileChooser fileOpen = new JFileChooser(newFile); // Create new file chooser dialog box
		ArrayList<Object> fileArray = new ArrayList<Object>(); // Holds array of strings read from data
		                                                           file

		int cancelOpen = fileOpen.showOpenDialog(baseFrame); // Show dialog box to open file

		if (cancelOpen == 0) { // Check to make sure file open dialog wasn't cancelled
			newFile = fileOpen.getSelectedFile(); // Get path and name of selected file
			lastOpenPath = newFile.getPath(); // Retain path of last file opened

			fileArray.add(newFile); // Add file path to file data array

			try { // Try to read data from file
				BufferedReader reader = new BufferedReader(new FileReader(newFile));
					// Read stream
				String line = null; // Initialize variable to get data
				while ((line = reader.readLine()) != null) { // Read in data until end of file
					String[] splitLine = line.split("\t"); // Split line string at tabs
					fileArray.add(splitLine); // Add split line strings to file array
				}
				reader.close(); // Close read stream
			} catch (Exception e) {
				JOptionPane.showMessageDialog(baseFrame, "Error reading file.");
					// Error dialog box
				e.printStackTrace();
			} // end try

			return fileArray; // Return array of data read from file
		} else {
			return null; // Return null value
		} // end if
	} // end loadFile()
```

```
/*-------------------------------------------------------------------------------------------
* Listener to register the goButton button press
* actionPerformed routine:
*         1. Check for data compatibility between the selected FCM and DSM files
*         2. Filter FCM matrix using imported functional model data
*         3. Build and filter the mini DSMs between the function pairings
*         4. Extract the valid component pairs from the filtered mini DSMs
*         *5. Build complete solutions that solve the entered function model
*         *6. Identify highest ranking solutions
*         7. Build GUI of result chains
*-------------------------------------------------------------------------------------------*/
private class goButtonListener implements ActionListener {
          public void actionPerformed(ActionEvent go) {

                    ArrayList<String[]> fcmArrayFiltered = new ArrayList<String[]>();
                              // Make array for filtered FCM

                    if (fmArray == null | fcmArray == null | dsmArray == null) {
                              // Make sure FM, FCM, and DSM files have been selected
                              JOptionPane.showMessageDialog(baseFrame, "Error: Please load all data files
                                        before continuing."); // Error dialog box
                    } else {
                              /* Check for FCM and DSM file compatibility--------------------------------- */
                              boolean match = false; // True if FCM component labels match DSM component
                                                       labels
                              String [] fcmString, dsmString; // Hold label rows for each matrix

                              fcmString = (String[]) fcmArray.get(fcmHeader); // Get label row from FCM
                              dsmString = (String[]) dsmArray.get(dsmHeader); // Get label row from DSM

                              match = Arrays.equals(fcmString,dsmString); // Component headers match?

                              if (match) { // If the FCM and DSM files are compatible
                                        fmLinks.clear(); // Clear list
                                        masterComponentList.clear(); // Clear component list

                                        /* Create subFunction link set------------------------------------ */
                                        for (int i = (fmHeader + 1); i < fmArray.size(); i++) {
                                                  // For each row in the FM matrix
                                                  String[] fmRow = (String[]) fmArray.get(i);
                                                  ArrayList<Integer> forConn = new ArrayList<Integer>();
                                                            // Array of forward connections
                                                  ArrayList<Integer> revConn = new ArrayList<Integer>();
                                                            // Array of reverse connections
                                                  Subfunction tempSub = new Subfunction(i, fmRow[0]);

                                                  for (int j = 1; j < fmRow.length; j++) {
                                                            /* Check for forward connections-------------- */
                                                            int cellForw = Integer.parseInt(fmRow[j]);
                                                                      // Get cell value
                                                            if (cellForw > 0) {
                                                                      forConn.add(j); // If cell is not zero,
                                                                                         add subfunction as a
                                                                                         forward connection
                                                            } // end if

                                                            /* Check for reverse connections-------------- */
                                                            String[] fmCol = (String[]) fmArray.get(j);
                                                            int cellRev = Integer.parseInt(fmCol[i]);
                                                                      // Get cell value
                                                            if (cellRev > 0) {
                                                                      revConn.add(j); // If cell is not zero,
                                                                                         add subfunction as a
                                                                                         reverse connection
                                                            } // end if
                                                  } // end for

                                                  tempSub.setForward(forConn);
                                                  tempSub.setReverse(revConn);
```

```
                fmLinks.add(tempSub);
        } // end for

        /* Filter the FCM matrix with the FM data------------------------- */
        String filter = null; // String to hold filter

        String[] headerRow = (String[]) fcmArray.get(0); // Get FCM header
                                                            row
        fcmArrayFiltered.add(headerRow); // Add header row to filtered FCM
        for (int ii = (fmHeader + 1); ii < fmArray.size(); ii++) {
                // Get filter from each row of the input FM
                String[] fmRow = (String[]) fmArray.get(ii);
                filter = fmRow[0];

                // For each row label in the FCM matrix
                boolean found = false; // Trigger to add empty FCM row
                for (int k = (fcmHeader + 1); k < fcmArray.size(); k++) {
                        String[] fcmRow = (String[]) fcmArray.get(k);
                                // Get FCM row

                        if (filter.equals(fcmRow[0])) {
                                // If filter matches FCM row
                                fcmArrayFiltered.add(fcmRow);
                                        // Add row to filtered FCM
                                found = true;
                                        // FCM contained filter value
                        } // end if
                } // end for

                if (!found) { // If row wasn't found for filter
                        String[] newRow = new String
                                [fcmString.length]; // Create a new
                                                       empty row
                        newRow[0] = filter; // Row header

                        for (int m = 1; m < newRow.length; m++) {
                                // Fill row with zeros
                                newRow[m] = "0"; // Indicates no
                                        component matches
                        } // end for

                        fcmArrayFiltered.add(newRow); // Add created row
                                to filtered matrix

                } // end if
        } // end for
```

```
******************Indentation on the following sections is shifted left 4 tabs to help avoid confusion*****************

        /* Build and filter pairwise DSM matrices------------------------- */
        for (int i = 0; i < fmLinks.size(); i++) {
                Subfunction functionForeward = fmLinks.get(i); // Cycle through each subfunction in the FM
                int functionID = functionForeward.getID(); // Get function id
                String functionLabel = functionForeward.getSub(); // Get function label
                ArrayList<Integer> connectedTo = functionForeward.getForward(); //Get forward connections

                for (int l = 0; l < connectedTo.size(); l++) {
                        int nextFunct = (Integer)connectedTo.get(l); // Get next connected function id
                        Subfunction functionReverse = fmLinks.get(nextFunct-1); // Point to next connected
                                                                                  subfunction

                        String[] func1 = (String[]) fcmArrayFiltered.get(functionID); // Get first row
                        String[] func2 = (String[]) fcmArrayFiltered.get(nextFunct); // Get second row

                        if (functionLabel.equals(func1[0])) { // Double check that functions are same

                                int[][] miniDSM = new int[func1.length-1][func2.length-1]; //Holds pairwise DSM

                                for (int m = 1; m < func1.length; m++) { // Build mini matrix
                                        int cellA = Integer.parseInt(func1[m]); // Get 1st cell
                                        String[] dsmRow = (String[]) dsmArray.get(m); // Get DSM row
                                        for (int n = 1; n < func2.length; n++) {
                                                int cellB = Integer.parseInt(func2[n]); // Get 2nd cell
                                                int dsmFilter; // Get filter value from DSM

                                                if (dsmRow[n].equals(" ") | dsmRow[n].equals("")) {
                                                        // Account for blank entry
                                                        dsmFilter = 0; // Makes blanks equal to zero
                                                } else { // Otherwise get DSM value
                                                        dsmFilter = Integer.parseInt(dsmRow[n]);
                                                } // end if

                                                if (dsmFilter != 0) { // If the filter is nonzero
                                                        miniDSM[m-1][n-1] = cellA*cellB*
                                                                (dsmFilter/dsmFilter); // Build DSM
                                                } else {
                                                        miniDSM[m-1][n-1] = cellA*cellB*dsmFilter;
                                                                // Build DSM
                                                } // end if
                                        } // end for
                                } // end for
                                functionForeward.addForwardDSM(miniDSM); // Add matrix to list

                                ArrayList reverseList = functionReverse.getReverse(); // Get list of reverse
                                                                                          connections
                                int ref = functionReverse.getReverseDSMs().size(); // How many DSMs are already
                                                                                        in the reverse list?

                                if (reverseList.get(ref) == (Integer)functionID) { // Check that the reverse
                                                                                        DSM is in the same order
                                                                                        as the reverse
                                                                                        connections list
                                        functionReverse.addReverseDSM(miniDSM); // Add this DSM as a reverse
                                                                                    DSM
                                } else { // Else print error information
                                        System.out.println("Error-----Reverse DSM add mismatch-----");
                                        System.out.println("  Reverse connection = " + reverseList.get(ref));
                                        System.out.println("     Reverse DSM functionID = " + functionID);
                                } // end if
                        } else {
                                System.out.println("Error-----Function mismatch-----");
                        } // end if
                } // end for loop
        } // end building DSMs

        /* Build choice lists--------------------------------------------------------------------------- */
        String[] dsmRow = (String[]) dsmArray.get(0); // Get DSM header row

        for (int i = 1; i < dsmRow.length; i++) {
                masterComponentList.add(dsmRow[i]); // Build component name index list
        } // end for

        // Check for in ports and out ports
        for (int j = 0; j < fmLinks.size(); j++) { // For each function in the model
                Subfunction function = fmLinks.get(j); // Get function
                ArrayList<Integer> forwardConns = function.getForward(); // Get forward connections
                ArrayList<Integer> reverseConns = function.getReverse(); // Get reverse connections


                // Check for in ports
                if (forwardConns.isEmpty()) { // If there are no forward connections
                        forwardConns.add(-20); // Add that the forward connection is an out port (-20)
                } // end if

                // Check for out ports
                if (reverseConns.isEmpty()) { // If there are no reverse connections
                        reverseConns.add(-10); // Add that the reverse connection is an in port (-10)
                } // end if

        } // end for

        // Build choice lists for all functions
        for (int j = 0; j < fmLinks.size(); j++) { // For each function in the model
```

```
Subfunction function = fmLinks.get(j); // Get next function
ArrayList forwardDSMList = function.getForwardDSMs(); // Get all forward connected DSMs
ArrayList reverseDSMList = function.getReverseDSMs(); // Get all reverse connected DSMs
ArrayList<ArrayList> forwardChoices = new ArrayList<ArrayList>(); // Initialize array to hold lists
                                                                     of choices for all forward DSM

for (int p = 0; p < forwardDSMList.size(); p++) { // For each forward DSM for this function
          ArrayList<ComponentInfo> thisForwCompList = new ArrayList<ComponentInfo>();
                    // Make list of choices for current forward DSM
          int[][] tempDSM = (int[][])forwardDSMList.get(p); // Get next DSM in the list

          for (int m = 0; m < tempDSM.length; m++) { // For each DSM row
                    int rowAddition = 0; // Initialize variable

                    for (int n = 0; n < tempDSM[0].length; n++) { // For each DSM column
                              rowAddition += tempDSM[m][n]; // Add row into a single variable
                    } // end for

                    if (rowAddition > 0) { // If there is a successful component pair in this row
                              ComponentInfo thisComponent = new ComponentInfo
                                        (masterComponentList.get(m)); // Make new component object
                              thisForwCompList.add(thisComponent); // Add choice to list
                    } // end if
          } // end for

          if (thisForwCompList.size() > 0) {
                    forwardChoices.add(thisForwCompList); // Add choice list to list of all forward
                                                             choices
          } // end if
} // end for

ArrayList<ArrayList> reverseChoices = new ArrayList<ArrayList>(); // Initialize array to hold lists
                                                                     of choices for all reverse DSM

for (int p = 0; p < reverseDSMList.size(); p++) { // For each reverse DSM for this function
          ArrayList<ComponentInfo> thisRevCompList = new ArrayList<ComponentInfo>();
                    // Make list of choices for current reverse DSM
          int[][] tempDSM = (int[][])reverseDSMList.get(p); // Get next DSM in the list

          for (int m = 0; m < tempDSM[0].length; m++) { // For each DSM column
                    int colAddition = 0; // Initialize variable

                    for (int n = 0; n < tempDSM.length; n++) { // For each DSM row
                              colAddition += tempDSM[n][m]; // Add column into a single variable
                    } // end for

                    if (colAddition > 0) { // If there is a successful component pair in this
                                              column
                              ComponentInfo thisComponent = new ComponentInfo
                                        (masterComponentList.get(m)); // Make new component object
                              thisRevCompList.add(thisComponent); // Add choice to list
                    } // end if
          } // end for

          if (thisRevCompList.size() > 0) {
                    reverseChoices.add(thisRevCompList); // Add choice list to list of all reverse
                                                           choices
          } // end if
} // end for

ArrayList<ComponentInfo> choiceList = new ArrayList<ComponentInfo>();
          // List of component choices to be set for current function
int numConnections = forwardChoices.size() + reverseChoices.size(); // Number of occurrences needed
                                                                       for each component

for (String nextComponent : masterComponentList) {
          int occurrences = 0; // Initialize variable to count number of component matches

          for (ArrayList<ComponentInfo> nextForwardList : forwardChoices) {
                    for (ComponentInfo nextForward : nextForwardList) {
                              if (nextForward.getComponent().equals(nextComponent)) {
                                        occurrences += 1; // Increment the number of occurrences
                                        } // end if
                                } // end for
                        } // end for
                        for (ArrayList<ComponentInfo> nextReverseList : reverseChoices) {
                                for (ComponentInfo nextReverse : nextReverseList) {
                                        if (nextReverse.getComponent().equals(nextComponent)) {
                                                occurrences += 1; // Increment the number of occurrences
                                        } // end if
                                } // end for
                        } // end for

                        if (occurrences == numConnections and numConnections != 0) { // If the component is a
match                                                                                   for all links
                                choiceList.add(new ComponentInfo(nextComponent)); // Add component to the
                                                                                  choice list
                        } // end if
                } // end for

                if (choiceList.isEmpty()) {
                        ComponentInfo noSolution = new ComponentInfo("?");
                        choiceList.add(noSolution); // Add question mark indicator
                        functionsWithoutSolutions.add(function.getID()); // Add function ID to list of functions
                                                                          without solutions
                } // end if
                function.setChoices(choiceList);
        } // end for

        /* Build component link lists------------------------------------------------------------------- */
        for (int j = 0; j < fmLinks.size(); j++) { // For each function in the model
                Subfunction function = fmLinks.get(j); // Get function
                ArrayList<int[][]> forwardDSMList = function.getForwardDSMs(); // Get all forward connected DSMs
                ArrayList<int[][]> reverseDSMList = function.getReverseDSMs(); // Get all reverse connected DSMs
                ArrayList<ComponentInfo> allChoices = function.getChoices(); // Get all component choices

                for (ComponentInfo nextChoice : allChoices) { // For each component choice
                        int componentIndex = masterComponentList.indexOf(nextChoice.getComponent()); // Get DSM
                                                                                                     index

                        if (componentIndex >= 0) {
                                for (int[][] tempDSM : forwardDSMList) { // For each forward DSM for this
                                                                          function
                                        ArrayList<String> foreLink = new ArrayList<String>();
                                                // Temp holder for fore links

                                        for (int n = 0; n < tempDSM[componentIndex].length; n++) {
                                                // For each DSM column in the index row
                                                if (tempDSM[componentIndex][n] != 0) {
                                                        // If there is a link between components

                                                        foreLink.add(masterComponentList.get(n));
                                                                // Add component to forward link list
                                                } // end if
                                        } // end for

                                        if (foreLink.isEmpty()) {
                                                foreLink.add("unknown");
                                        } // end if

                                        nextChoice.addForwardLinks(foreLink); // Add array to link list
                                } // end for

                                for (int[][] tempDSM : reverseDSMList) { // For each reverse DSM for this
                                                                          function
                                        ArrayList<String> revLink = new ArrayList<String>();
                                                // Temp holder for reverse links

                                        for (int n = 0; n < tempDSM.length; n++) { // For each DSM row in the
                                                                                    index column
                                                if (tempDSM[n][componentIndex] != 0) {
                                                        // If there is a link between components
                                                        revLink.add(masterComponentList.get(n));
                                                                // Add component to reverse link list
                        } // end if
                    } // end for

                    if (revLink.isEmpty()) {
                        revLink.add("unknown");
                    } // end if

                    nextChoice.addReverseLinks(revLink); // Add array to link list
                } // end for
            } else {
                ArrayList<String> foreLink = new ArrayList<String>(); // Temp holder for fore
                                                                        links
                foreLink.add("unknown");
                nextChoice.addForwardLinks(foreLink); // Add array to link list

                ArrayList<String> revLink = new ArrayList<String>(); // Temp holder for reverse
                                                                       links
                revLink.add("unknown");
                nextChoice.addReverseLinks(revLink); // Add array to link list
            } // end if
        } // end for

        ArrayList<Integer> forwardConns = function.getForward(); // Get forward connections
        ArrayList<Integer> reverseConns = function.getReverse(); // Get reverse connections
        ArrayList<ComponentInfo> choiceList = function.getChoices(); // Get list of choices

        if (forwardConns.contains(-20)) { // If there are no forward connections because of an out port
            for (ComponentInfo choice : choiceList) { // Set all component forward links as system
                                                         outs
                ArrayList<String> foreport = new ArrayList<String>();
                ArrayList<ArrayList<String>> foreports = new ArrayList<ArrayList<String>>();
                foreport.add("out");
                foreports.add(foreport);
                choice.setForwardLinks(foreports);
            } // end for
        } // end if

        if (reverseConns.contains(-10)) { // If there are no reverse connections because of an in port
            for (ComponentInfo choice : choiceList) { // Set all component forward links as system
                                                         ins
                ArrayList<String> aftport = new ArrayList<String>();
                ArrayList<ArrayList<String>> aftports = new ArrayList<ArrayList<String>>();
                aftport.add("in");
                aftports.add(aftport);
                choice.setReverseLinks(aftports);
            } // end for
        } // end if
    } // end for
************************************************End shifted indentation************************************************
                    buildOutputGUI();
                } else {
                    JOptionPane.showMessageDialog(baseFrame, "Error: FCM and DSM matrices
                                                  are not compatible.\nPlease choose
                                                  compatible files.");
                                                  // Error dialog box
                } // end if
            } // end if
        } // end actionPerformed
    } // end goButtonListener

    /*------------------------------------------------------------------------------------------
     * Builds the graphical user interface for the user input into the concept generation program
     *------------------------------------------------------------------------------------------*/
    private void buildOutputGUI()  {

        fullCompArray.clear(); // Clear button array
        removeButtons.clear(); // Clear button array
        selectedFields.clear(); // Clear textfield array

        if (resultFrame != null) { // If an old result frame exists
            resultFrame.dispose(); // Get rid of old result frame before generating a new one
        }
```

```
/* Define and Initialize variables ---------------------------------------------------------*/
resultFrame = new JFrame();
resultBackground = new JPanel();

JPanel step4Panel = new JPanel(), headerPanel = new JPanel(), resultsPanel = new JPanel();
JTextField selectedComponentLabel = new JTextField(), blankLabel = new JTextField();
JTextField subFunctionLabel = new JTextField(), componentLabel = new JTextField();
JScrollPane resultsScrollPane = new JScrollPane();

/* resultFrame ----------------------------------------------------------------------------*/
{
          resultFrame.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
          resultFrame.setResizable(true);
          Container resultFrameContentPane = resultFrame.getContentPane();
          resultFrameContentPane.setLayout(new BorderLayout(10, 10));

          /* resultBackground ------------------------------------------------------------------*/
          {
                    resultBackground.setBorder(new EmptyBorder(10, 10, 10, 10));
                    resultBackground.setBackground(lightblue);
                    resultBackground.setLayout(new BoxLayout(resultBackground, BoxLayout.Y_AXIS));

                    /* step4Panel ---------------------------------------------------------------*/
                    {
                              step4Panel.setBorder(new CompoundBorder(
                                        new TitledBorder("Step 4: Build and evaluate conceptual
                                                          solutions."),
                                        new EmptyBorder(5, 5, 5, 5)));
                              step4Panel.setBackground(lightblue);
                              step4Panel.setLayout(new BorderLayout());

                              /* headerPanel ----------------------------------------------------*/
                              {
                                        headerPanel.setMinimumSize(new Dimension(427, 35));
                                        headerPanel.setMaximumSize(new Dimension(427, 35));
                                        headerPanel.setLayout(new BoxLayout(headerPanel,
                                                  BoxLayout.X_AXIS));

                                        /* selectedComponentLabel -------------------------------*/
                                        selectedComponentLabel.setText("Selected");
                                        selectedComponentLabel.setBorder(new EmptyBorder(0,0,0,0));
                                        selectedComponentLabel.setHorizontalAlignment
                                                  (SwingConstants.CENTER);
                                        selectedComponentLabel.setMaximumSize(new Dimension(155,
                                                                                    35));
                                        selectedComponentLabel.setPreferredSize(new Dimension(155,
                                                                                    35));
                                        selectedComponentLabel.setMinimumSize(new Dimension(155,
                                                                                    35));
                                        selectedComponentLabel.setBackground(darkblue);
                                        selectedComponentLabel.setFocusable(false);
                                        selectedComponentLabel.setEditable(false);
                                        headerPanel.add(selectedComponentLabel);

                                        /* blankLabel -------------------------------------------*/
                                        blankLabel.setBorder(new EmptyBorder(0, 0, 0, 0));
                                        blankLabel.setHorizontalAlignment(SwingConstants.CENTER);
                                        blankLabel.setMaximumSize(new Dimension(80, 35));
                                        blankLabel.setPreferredSize(new Dimension(80, 35));
                                        blankLabel.setMinimumSize(new Dimension(80, 35));
                                        blankLabel.setBackground(darkblue);
                                        blankLabel.setFocusable(false);
                                        blankLabel.setEditable(false);
                                        headerPanel.add(blankLabel);

                                        /* subFunctionLabel -------------------------------------*/
                                        subFunctionLabel.setText("Subfunctions");
                                        subFunctionLabel.setBorder(new EmptyBorder(0, 0, 0, 0));
                                        subFunctionLabel.setHorizontalAlignment
                                                            (SwingConstants.CENTER);
                                        subFunctionLabel.setMaximumSize(new Dimension(155, 35));
                    subFunctionLabel.setMinimumSize(new Dimension(155, 35));
                    subFunctionLabel.setPreferredSize(new Dimension(155, 35));
                    subFunctionLabel.setBackground(darkblue);
                    subFunctionLabel.setFocusable(false);
                    subFunctionLabel.setEditable(false);
                    headerPanel.add(subFunctionLabel);

                    /* componentLabel ---------------------------------------*/
                    componentLabel.setText("Component Solutions -->");
                    componentLabel.setBorder(new EmptyBorder(0, 0, 0, 0));
                    componentLabel.setBackground(darkblue);
                    componentLabel.setFocusable(false);
                    componentLabel.setEditable(false);
                    headerPanel.add(componentLabel);
          }
          step4Panel.add(headerPanel, BorderLayout.NORTH);

          /* resultsScrollPane ---------------------------------------------*/
          {
                    int boxHeight = 30, boxWidth = 150;
                    int spaceTall = 80;
                    int scrollPaneHeight = 400, scrollPaneWidth = 1000;
                    int scrollPanelHeight = spaceTall*(fmArray.size()-1),
                        scrollPanelWidth = 1000;
                    int compScrollPaneHeight = spaceTall;
                    int compScrollPaneWidth = 600;
                        rowPanelHeight = spaceTall, rowPanelWidth = 1000;

                    resultsScrollPane.setHorizontalScrollBarPolicy
                              (ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);
                    resultsScrollPane.setMinimumSize(new Dimension
                              (scrollPaneWidth, scrollPaneHeight));
                    resultsScrollPane.setMaximumSize(new Dimension
                              (scrollPaneWidth, scrollPaneHeight));
                    resultsScrollPane.setPreferredSize(new Dimension
                              (scrollPaneWidth, scrollPaneHeight));

******************Indentation on the following sections is shifted left 4 tabs to help avoid confusion*****************

                    /* resultsPanel ----------------------------------------*/
                    {
                              resultsPanel.setMinimumSize(new Dimension(scrollPanelWidth, scrollPanelHeight));
                              resultsPanel.setMaximumSize(new Dimension(scrollPanelWidth, scrollPanelHeight));
                              resultsPanel.setPreferredSize(new Dimension(scrollPanelWidth, scrollPanelHeight));
                              resultsPanel.setLayout(new BoxLayout(resultsPanel, BoxLayout.Y_AXIS));

                              /* Populate panels ----------------------------*/
                              for (int i = 0; i < fmLinks.size(); i++) {
                                        Subfunction tempFunction = fmLinks.get(i);

                                        /* Full row panel -------------------*/
                                        JPanel rowPanel = new JPanel();
                                        rowPanel.setMinimumSize(new Dimension(rowPanelWidth, rowPanelHeight));
                                        rowPanel.setMaximumSize(new Dimension(rowPanelWidth, rowPanelHeight));
                                        rowPanel.setPreferredSize(new Dimension(rowPanelWidth, rowPanelHeight));
                                        rowPanel.setLayout(new BoxLayout(rowPanel, BoxLayout.X_AXIS));

                                        {
                                                  /* Component selections ---*/
                                                  JTextField selectedComponent = new JTextField();
                                                  selectedComponent.setBackground(Color.white);
                                                  selectedComponent.setAutoscrolls(false);
                                                  selectedComponent.setBorder(new BevelBorder(BevelBorder.LOWERED));
                                                  selectedComponent.setHorizontalAlignment(SwingConstants.CENTER);
                                                  selectedComponent.setMinimumSize(new Dimension(boxWidth, boxHeight));
                                                  selectedComponent.setMaximumSize(new Dimension(boxWidth, boxHeight));
                                                  selectedComponent.setPreferredSize(new Dimension(boxWidth,
                                                                                                  boxHeight));
                                                  selectedComponent.setText("None");
                                                  selectedComponent.setEditable(false);
                                                  selectedFields.add(selectedComponent);
```

```java
/* Remove buttons -------------------------------------------------*/
JButton removeButton = new JButton();
removeButton.setText("Remove");
removeButton.addActionListener(new removeButtonListener());
removeButtons.add(removeButton);

/* Subfunction fields ---------------------------------------------*/
JPanel subFunctionPanel = new JPanel();
subFunctionPanel.setBorder(new EmptyBorder(20,5,5,0));
subFunctionPanel.setLayout(new BoxLayout(subFunctionPanel,
          BoxLayout.Y_AXIS));
JTextArea subFunction = new JTextArea();
subFunction.setAutoscrolls(false);
subFunction.setLineWrap(true);
subFunction.setWrapStyleWord(true);
subFunction.setMinimumSize(new Dimension(boxWidth, boxHeight*2));
subFunction.setMaximumSize(new Dimension(boxWidth, boxHeight*2));
subFunction.setOpaque(false);
subFunction.setText(tempFunction.getSub());
subFunction.setPreferredSize(new Dimension(boxWidth, boxHeight*2));
subFunction.setEditable(false);
subFunctionPanel.add(subFunction);

/* compScrollPane -------------------------------------------------*/
JScrollPane compScrollPane = new JScrollPane();
compScrollPane.setVerticalScrollBarPolicy
          (ScrollPaneConstants.VERTICAL_SCROLLBAR_NEVER);
compScrollPane.setMinimumSize(new Dimension(compScrollPaneWidth,
          compScrollPaneHeight));
compScrollPane.setMaximumSize(new Dimension(compScrollPaneWidth,
          compScrollPaneHeight));
compScrollPane.setPreferredSize(new Dimension(compScrollPaneWidth,
          compScrollPaneHeight));

/* Component fields -----------------------------------------------*/
ArrayList<JButton> componentArray = new ArrayList<JButton>();
ArrayList components = tempFunction.getChoices();
int numComps = components.size();
int compPanelHeight = boxHeight,
    compPanelWidth = boxWidth * numComps;

/* Component row panel --------------------------------------------*/
JPanel compRowPanel = new JPanel();
compRowPanel.setMinimumSize(new Dimension(compPanelWidth,
          compPanelHeight));
compRowPanel.setMaximumSize(new Dimension(compPanelWidth,
          compPanelHeight));
compRowPanel.setPreferredSize(new Dimension(compPanelWidth,
          compPanelHeight));
compRowPanel.setLayout(new BoxLayout(compRowPanel,
          BoxLayout.X_AXIS));

for (int j = 0; j < components.size(); j++) {
          JButton component = new JButton();
          ComponentInfo nextComp = (ComponentInfo) components.get(j);
                    // Get next component in list
          component.setText(nextComp.getComponent());
          component.setHorizontalAlignment(SwingConstants.CENTER);
          component.setCursor(Cursor.getPredefinedCursor
                    (Cursor.HAND_CURSOR));
          component.setMinimumSize(new Dimension(95, 35));
          component.setMaximumSize(new Dimension(150, 35));
          component.addActionListener(new selectComponentListener());
          compRowPanel.add(component);
          componentArray.add(component);
}
compScrollPane.setViewportView(compRowPanel);
fullCompArray.add(componentArray);
rowPanel.add(selectedComponent);
rowPanel.add(removeButton);
rowPanel.add(subFunctionPanel);
rowPanel.add(compScrollPane);
```

```
                                        }
                                        resultsPanel.add(rowPanel);
                              } // end for loop
                    }

**********************************************End shifted indentation**********************************************
                                                            resultsScrollPane.setViewportView(resultsPanel);
                                                  }
                                                  step4Panel.add(resultsScrollPane, BorderLayout.CENTER);
                                        }
                                        resultBackground.add(step4Panel);
                              }
                              resultFrameContentPane.add(resultBackground, BorderLayout.CENTER);
                              resultFrame.pack();
                              resultFrame.setLocationRelativeTo(resultFrame.getOwner());
                              resultFrame.setVisible(true);
                    } // end resultsFrame
          } // end buildOutputGUI()

          /*-----------------------------------------------------------------------------------------------
          * Listener to register any component selection click:
          *         1. Determines which component was selected
          *         2. Deselects other components for that function
          *         3. Updates the results displayed
          *-----------------------------------------------------------------------------------------------*/
          private class selectComponentListener implements ActionListener {
                    public void actionPerformed(ActionEvent select) {
                              Object source = select.getSource(); // Determines button selection that triggered the
                                                                      listener
                              int functionNum = -1; // Initialize function number
                              String selectedComponent = null; // Initialize selected component

                              // Determine which component button was selected
                              for (ArrayList<JButton> compButtonList : fullCompArray) {
                                        for (JButton compButton : compButtonList) {
                                                  if (compButton == source) {
                                                            functionNum = fullCompArray.indexOf(compButtonList);
                                                            JTextField updateSelect = selectedFields.get(functionNum);
                                                            selectedComponent = compButton.getText();
                                                            updateSelect.setText(selectedComponent);
                                                  } // end if
                                        } // end for
                              } // end for

                              // Deselect other components for that function
                              ArrayList<JButton> desButtonList = fullCompArray.get(functionNum);

                              for (JButton desButton : desButtonList) { // For each component choice for this function
                                        if (desButton.getText() != selectedComponent) { // If the component is not the
                                                                                          one that was selected
                                                  desButton.setEnabled(false); // Deactivate the button
                                        } // end if
                              } // end for

                              updateResults(); // Refresh output to eliminate incompatible choices

                    } // actionPerformed
          } // end componentSelectedListener

          /*-----------------------------------------------------------------------------------------------
          * Listener to register any remove button activation:
          *         1. Determines which textfield was selected
          *         2. Reactivates other components for that function
          *         3. Updates the results displayed
          *-----------------------------------------------------------------------------------------------*/
          private class removeButtonListener implements ActionListener {
                    public void actionPerformed(ActionEvent remove) {
                              Object source = remove.getSource(); // Determines textfield selection that triggered the
                                                                      listener

                              int functionNum = -1; // Initialize function number
                    // Determine which remove button was selected
                    for (JButton removeButton : removeButtons) {
                            if (removeButton == source) {
                                    functionNum = removeButtons.indexOf(removeButton);
                                            // Get index of function that had the component removed
                                    JTextField updateSelect = selectedFields.get(functionNum);
                                            // Get text field for selected component
                                    updateSelect.setText("None"); // Update text field to remove
                                                                  component choice
                            } // end if
                    } // end for

                    for (JTextField nextSelected : selectedFields) { // For each selected text field
                            if (nextSelected.getText().equalsIgnoreCase("None")) {
                                    // If there is no component selected
                                    int selectedIndex = selectedFields.indexOf(nextSelected);
                                            // Get index of current selection text field
                                    ArrayList<JButton> nextComponentList = fullCompArray.get
                                            (selectedIndex); // Get corresponding component button list

                                    for (JButton nextComponent : nextComponentList) {
                                            // For each component GUI button
                                            nextComponent.setEnabled(true); // Activate button
                                    } // end for
                            } // end if
                    } // end for

                    updateResults(); // Refresh output to eliminate incompatible choices

            } // end actionPerformed
    } // end remove Button Listener

    /*-------------------------------------------------------------------------------------------
     * Updates the graphical user interface for the interactive output frame by eliminating
     * incompatible choices.
     *-------------------------------------------------------------------------------------------*/
    private void updateResults() {
            boolean change = false;
            do {
                    change = false;

                    for (int i = 0; i < fmLinks.size(); i++) { // For each subfunction
                            Subfunction f = fmLinks.get(i); // Get next subfunction
                            ArrayList<ComponentInfo> choicesForF = f.getChoices();
                                    // Get list of component choices for this subfunction
                            ArrayList<Integer> fConnections = f.getForward(); // Get forward connected
                                                                               functions
                            ArrayList<Integer> aConnections = f.getReverse(); // Get reverse connected
                                                                               functions
                            JTextField selectedField = selectedFields.get(i);
                                    // Get selected component text field for this function
                            ArrayList<JButton> componentButtonList = fullCompArray.get(i);
                                    // GUI buttons for this function's components
                            ArrayList<String> activeValidForeComps = new ArrayList<String>();
                                    // List of active valid components from forward connections
                            ArrayList<String> activeValidAftComps = new ArrayList<String>();
                                    // List of active valid components from reverse connections

                            if (selectedField.getText().equalsIgnoreCase("None")) {
                                    // If a component has not yet been selected for the current component
                                    for (int fConnectID : fConnections) { // For each forward connected
                                                                           function
                                            if (fConnectID > 0) { // If the forward connection is not
                                                                     an out port
                                                    ArrayList<JButton> nextForwardButtons =
                                                            fullCompArray.get(fConnectID-1);
                                                            // Get GUI button array for next
                                                               forward connected function

*****************Indentation on the following sections is shifted left 4 tabs to help avoid confusion*****************

                    for (JButton nextForwardButton :
                            nextForwardButtons) { // For each GUI button

                            if (nextForwardButton.isEnabled()) { // If the button is still active
                                    String componentText = new String();

                                    if (nextForwardButton.getText() == "?") { // If forward component is
                                                                               unknown
                                            componentText = "unknown"; // Set text to "unknown"
                                    } else {
                                            componentText = nextForwardButton.getText();
                                                    // Set text to button text
                                    } // end if

                                    if (!activeValidForeComps.contains(componentText)) {
                                            // If component name is not already in the list
                                            activeValidForeComps.add(componentText);
                                                    // Add the active valid component to the list
                                    } // end if
                            } // end if
                    } // end for
            } // end if
    } // end for

    for (int aConnectID : aConnections) { // For each reverse connected function
            if (aConnectID > 0) { // If the reverse connection is not an in port
                    ArrayList<JButton> nextReverseButtons = fullCompArray.get(aConnectID-1);
                            // Get GUI button array for next reverse connected function

                    for (JButton nextReverseButton : nextReverseButtons) { // For each GUI button
                            if (nextReverseButton.isEnabled()) { // If the button is still active
                                    String componentText = new String();

                                    if (nextReverseButton.getText() == "?") { // If forward component is
                                                                               unknown
                                            componentText = "unknown"; // Set text to "unknown"
                                    } else {
                                            componentText = nextReverseButton.getText();
                                                    // Set text to button text
                                    } // end if

                                    if (!activeValidAftComps.contains(componentText)) {
                                            // If component name is not already in the list
                                            activeValidAftComps.add(componentText);
                                                    // Add the active valid component to the list
                                    } // end if
                            } // end if
                    } // end for
            } // end if
    } // end for

    for (int j = 0; j < choicesForF.size(); j++) { // For each component choice in the current function's choice
                                                     list
            ComponentInfo choice = choicesForF.get(j); // Get next component choice
            JButton choiceButton = componentButtonList.get(j); // Get component GUI button
            ArrayList<ArrayList<String>> foreLinks = choice.getForwardLinks(); // Get the component's forward
                                                                                links
            ArrayList<ArrayList<String>> aftLinks = choice.getReverseLinks(); // Get the component's reverse
                                                                               links

            boolean noValidForeLinks = true;
            boolean outport = false;
            for (ArrayList<String> nextForeLinks : foreLinks) { // For each valid forward component link list
                                                                  for current component
                    for (String nextForeLink : nextForeLinks) { // For each valid forward component for
                                                                  current component
                            for (String validForeLink : activeValidForeComps) {
                                    // Compare against each active valid forward component
                                    if (nextForeLink.equalsIgnoreCase(validForeLink)) {
                                            // If there is a matching active link
                                            noValidForeLinks = false; // Change boolean value
                                    } // end if
                            } // end for
```

```
                                        if (nextForeLink == "out") { // Check for an out port
                                                  outport = true; // Change boolean value
                                        } // end if

                                        if (nextForeLink == "unknown" and nextForeLinks.size() == 1) {
                                                  noValidForeLinks = false; // Change boolean value
                                        }
                              } // end for
                    } // end for

                    if (foreLinks.size() == 1 and noValidForeLinks and outport) { // If there is only one output, no
valid
                                                                                  links, and the output is an out port
                              noValidForeLinks = false; // Change boolean value
                    }

                    boolean noValidAftLinks = true;
                    boolean inport = false;
                    for (ArrayList<String> nextAftLinks : aftLinks) { // For each valid reverse component link list for
                                                                         current component
                              for (String nextAftLink : nextAftLinks) { // For each valid reverse component for current
                                                                           component
                                        for (String validAftLink : activeValidAftComps) { // Compare against each
                                                                                            active valid reverse
                                                                                            component
                                                  if (nextAftLink.equalsIgnoreCase(validAftLink)) { // If there is a
                                                                                                      matching active
                                                                                                      link
                                                            noValidAftLinks = false; // Change boolean value
                                                  } // end if
                                        } // end for
                                        if (nextAftLink == "in") { // Check for an in port
                                                  inport = true; // Change boolean value
                                        } // end if

                                        if (nextAftLink == "unknown" and nextAftLinks.size() == 1) {
                                                  noValidAftLinks = false; // Change boolean value
                                        }
                              } // end for
                    } // end for

                    if (aftLinks.size() == 1 and noValidAftLinks and inport) { // If there is only one input, no valid
                                                                               links, and the input is an in port
                              noValidAftLinks = false; // Change boolean value
                    }

                    if ((noValidForeLinks | noValidAftLinks)andchoiceButton.getText() !="?" and choiceButton.isEnabled
()) {
                              // If the current component has no active forward or reverse links
                              choiceButton.setEnabled(false); // Deactivate the component
                              //change = true; // Indicate a change has been made
                    } // end if

          } // end for
} // end if

************************************************End shifted indentation************************************************

                              } // end for
                    } while (change);
          } // end updateResults

} // end ConGenV2_0
```

---------------------------------------end ‘ConGen.java’ code ------------------------------------

-----------------------------------start ‘Subfunction.java’ code -----------------------------------

```
package edu.umr.ide;

import java.util.ArrayList;

public class Subfunction {
	int id = -1; // Unique assigned to the subfunction
	String functionLabel = null; // Subfunction label from functional model
	ArrayList<Integer> forwardConnect = new ArrayList<Integer>(); // Other subfunction ids that this subfunction
	                                                              connects to
	ArrayList<Integer> reverseConnect = new ArrayList<Integer>(); // Other subfunction ids that are connected to
	                                                              this subfunction
	ArrayList<int[][]> forwardDSMs = new ArrayList<int[][]>(); // Collection of forward DSMs for this function
	ArrayList<int[][]> reverseDSMs = new ArrayList<int[][]>(); // Collection of reverse DSMs for this function
	ArrayList<ComponentInfo> choices = new ArrayList<ComponentInfo>(); // Component choices to fulfil this
	                                                                   function

	public Subfunction() {
		id = -1; // Set id
		functionLabel = null; // Set label
	}
	public Subfunction(int functID, String funct) {
		id = functID; // Set id
		functionLabel = funct; // Set label
	}

	public int getID() {
		return id;
	}

	public String getSub() {
		return functionLabel;
	}

	public void setForward(ArrayList<Integer> forward) {
		forwardConnect = forward;
	}

	public ArrayList<Integer> getForward() {
		return forwardConnect;
	}

	public void setReverse(ArrayList<Integer> reverse) {
		reverseConnect = reverse;
	}

	public ArrayList<Integer> getReverse() {
		return reverseConnect;
	}

	public void setForwardDSMs(ArrayList<int[][]> dsm) {
		forwardDSMs = dsm;
	}

	public ArrayList<int[][]> getForwardDSMs() {
		return forwardDSMs;
	}

	public void addForwardDSM(int[][] singleDSM) {
		forwardDSMs.add(singleDSM); // Add single DSM to array
	}

	public void setReverseDSMs(ArrayList<int[][]> dsm) {
		reverseDSMs = dsm;
	}

	public ArrayList<int[][]> getReverseDSMs() {
		return reverseDSMs;
	}

	public void addReverseDSM(int[][] singleDSM) {
		reverseDSMs.add(singleDSM); // Add single DSM to array
	}

	public ArrayList<ComponentInfo> getChoices() {
		return choices;
	}

	public void setChoices(ArrayList<ComponentInfo> newChoices) {
		choices = newChoices;
	}

	public void addChoice(ComponentInfo choiceToAdd) {
		choices.add(choiceToAdd);
	}

	public void removeChoice(int removeIndex) {
		choices.remove(removeIndex);
	}
} // end Subfunction
```

-------------------------------------end ‘Subfunction.java’ code -----------------------------------

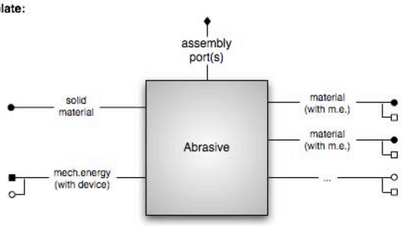
----------------------------------start ‘ComponentInfo.java’ code ----------------------------------
```
package edu.umr.ide;

import java.util.ArrayList;

public class ComponentInfo {
	String componentName; // Component label
	ArrayList<ArrayList<String>> forwardLinks = new ArrayList<ArrayList<String>>();
		// Components that this component is forward linked to
	ArrayList<ArrayList<String>> reverseLinks = new ArrayList<ArrayList<String>>();
		// Components that this component is reverse linked to

	// Establish constructors
	public ComponentInfo() {
		componentName = "?"; // Default component name
	}

	public ComponentInfo(String label) {
		componentName = label; // Set component name to the label input
	}


	// Methods for retrieving general component information
	public String getComponent() {
		return componentName; // Return the name of the component
	}

	public ArrayList<ArrayList<String>> getForwardLinks() {
		return forwardLinks; // Return the array of components that are forward linked this component
	}

	public ArrayList<ArrayList<String>> getReverseLinks() {
		return reverseLinks; // Return the array of components that are forward linked this component
	}


	// Methods for establishing general component information
	public void setComponent(String label) {
		componentName = label; // Sets the name of the component
	}

	public void setForwardLinks(ArrayList<ArrayList<String>> links) {
		forwardLinks = links; // Set the forward links for this component
	}

	public void setReverseLinks(ArrayList<ArrayList<String>> links) {
		reverseLinks = links; // Set the forward links for this component
	}


	// Methods for adding to existing component information
	public void addForwardLinks(ArrayList<String> forelinks) {
		forwardLinks.add(forelinks); // Add to forwardLinks list
	}

	public void addReverseLinks(ArrayList<String> revlinks) {
		reverseLinks.add(revlinks); // Add to reverseLinks list
	}
} // end ComponentInfo
```
----------------------------------end ‘ComponentInfo.java’ code -----------------------------------

APPENDIX B

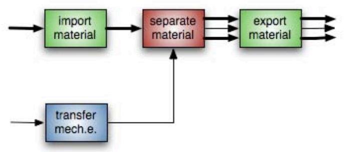
Component Templates

Component Name: Abrasive

Port Template:



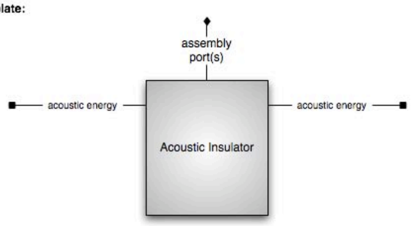
Function Template:



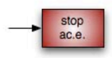
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Acoustic Insulator

Port Template:



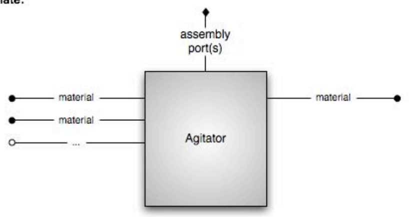
Function Template:



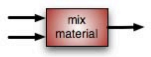
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Agitator

Port Template:



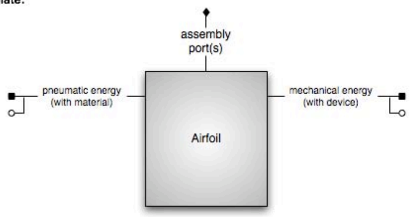
Function Template:



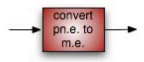
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Airfoil

Port Template:


assembly port(s)
pneumatic energy (with material)
Airfoil
mechanical energy (with device)

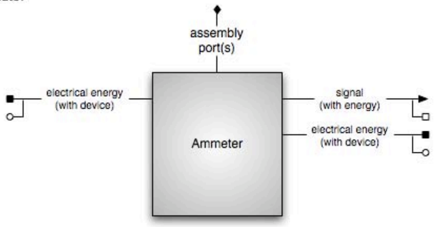
Function Template:



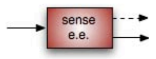
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Ammeter

Port Template:



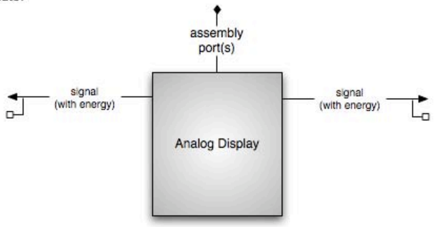
Function Template:



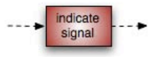
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Analog Display

Port Template:



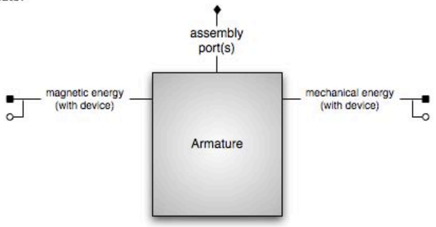
Function Template:



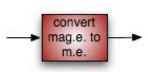
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Armature

Port Template:



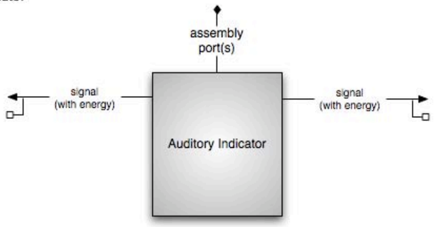
Function Template:



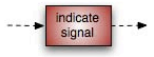
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Auditory Indicator

Port Template:



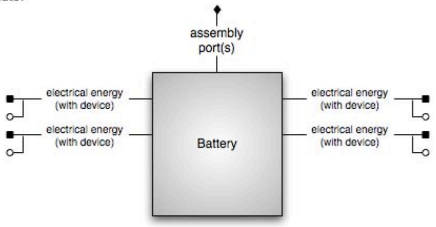
Function Template:



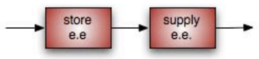
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Battery

Port Template:


assembly port(s)
electrical energy (with device)
electrical energy (with device)
Battery
electrical energy (with device)
electrical energy (with device)

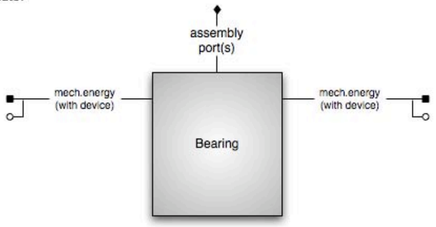
Function Template:



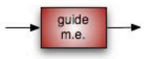
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Bearing

Port Template:



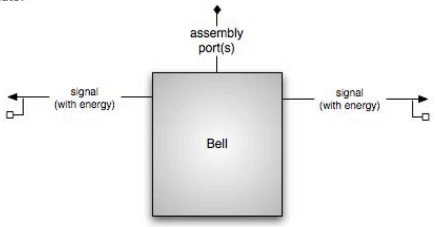
Function Template:



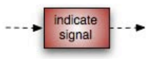
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Bell

Port Template:



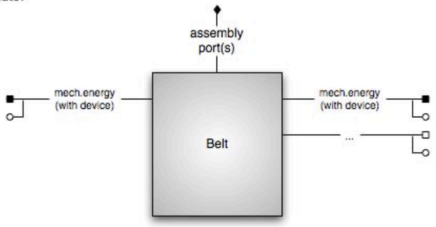
Function Template:



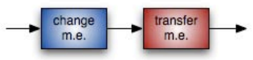
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Belt

Port Template:



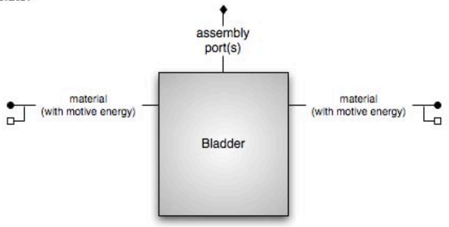
Function Template:



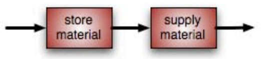
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Bladder

Port Template:



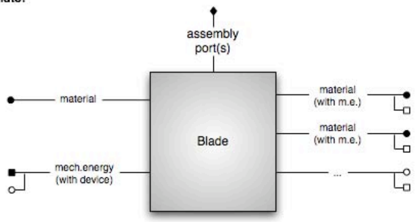
Function Template:



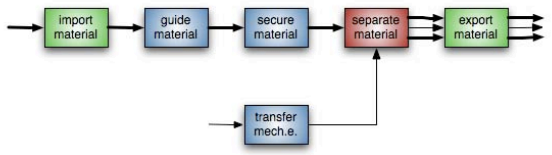
Key: 🟦 Sometimes included 🟩 Usually included 🟥 Required (Black Box Function(s))

Component Name: Blade

Port Template:



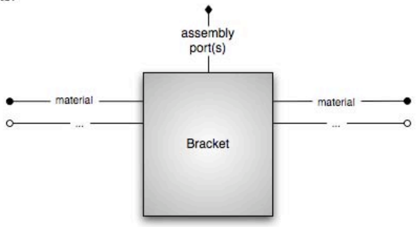
Function Template:



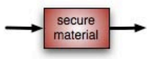
Key: 🟦 Sometimes included 🟩 Usually included 🟥 Required (Black Box Function(s))

Component Name: Bracket

Port Template:



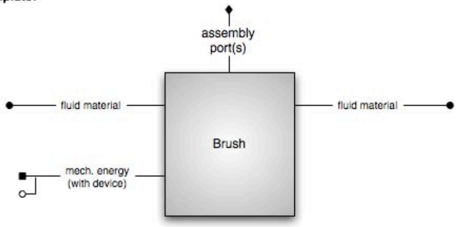
Function Template:



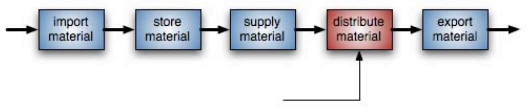
Key: 🟦 Sometimes included 🟩 Usually included 🟥 Required (Black Box Function(s))

Component Name: Brush

Port Template:



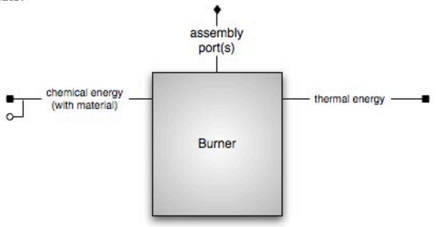
Function Template:



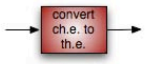
Key: 🟦 Sometimes included 🟩 Usually included 🟥 Required (Black Box Function(s))

Component Name: Burner

Port Template:



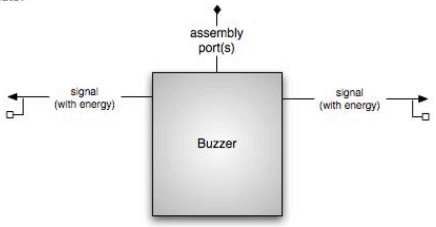
Function Template:



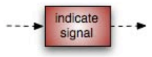
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Buzzer

Port Template:



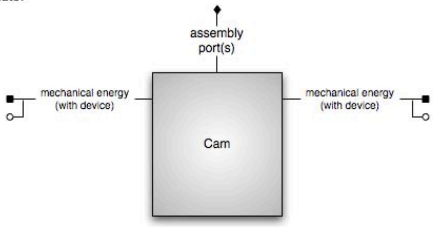
Function Template:



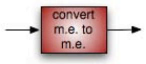
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Cam

Port Template:



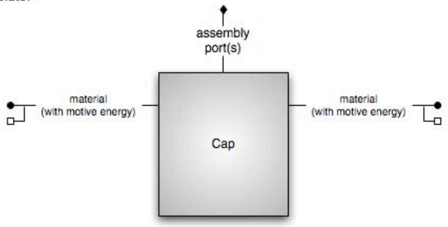
Function Template:



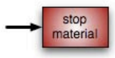
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Cap

Port Template:



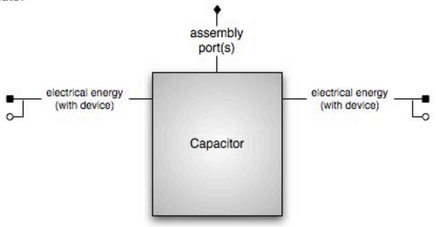
Function Template:



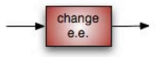
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Capacitor

Port Template:


assembly port(s)
electrical energy (with device)
Capacitor
electrical energy (with device)

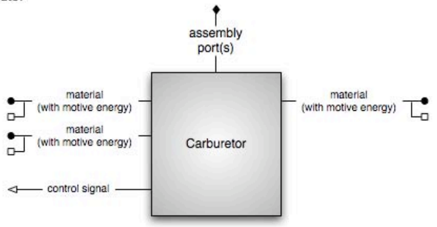
Function Template:



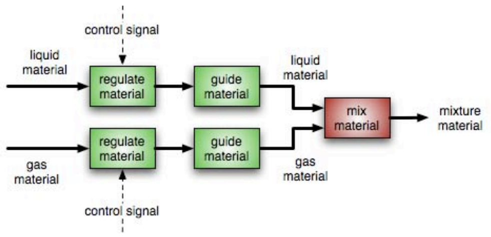
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Carburetor

Port Template:


assembly port(s)
material (with motive energy)
material (with motive energy)
control signal
Carburetor
material (with motive energy)

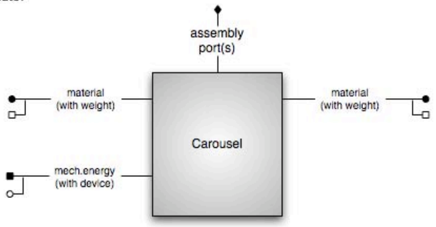
Function Template:



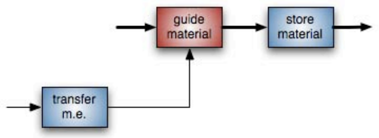
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Carousel

Port Template:



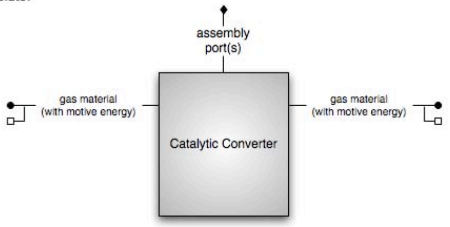
Function Template:



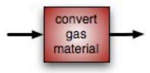
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Catalytic Converter

Port Template:


assembly port(s)
gas material (with motive energy)
Catalytic Converter
gas material (with motive energy)

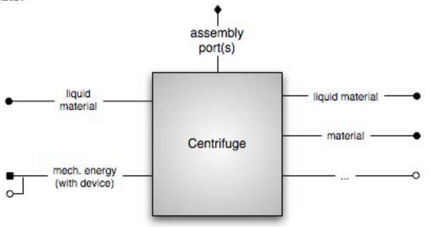
Function Template:



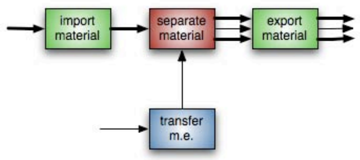
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Centrifuge

Port Template:



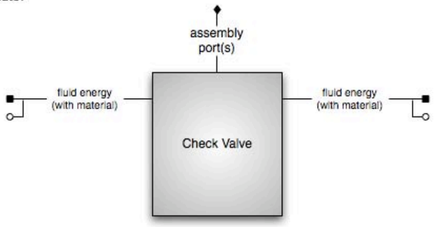
Function Template:



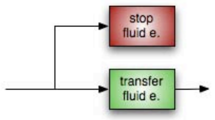
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Check Valve

Port Template:



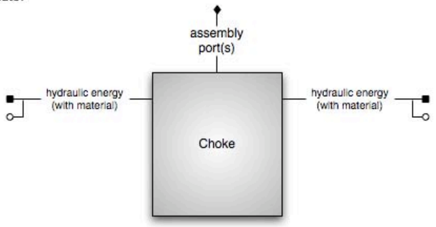
Function Template:



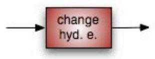
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Choke

Port Template:


assembly port(s)
hydraulic energy (with material)
Choke
hydraulic energy (with material)

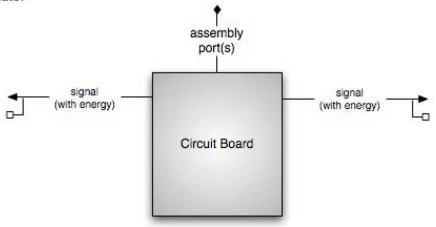
Function Template:



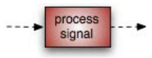
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Circuit Board

Port Template:



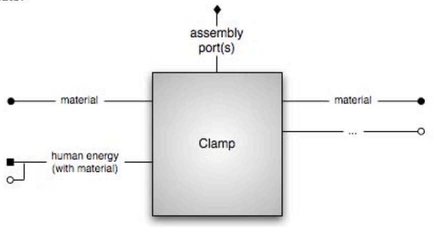
Function Template:



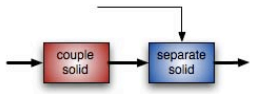
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Clamp

Port Template:



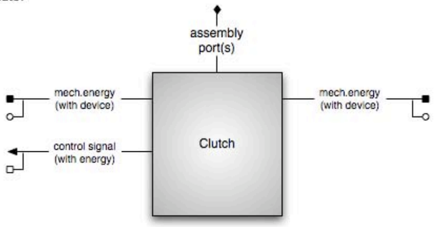
Function Template:



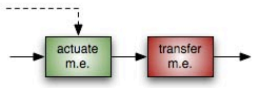
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Clutch

Port Template:


assembly port(s)
mech.energy (with device)
mech.energy (with device)
control signal (with energy)
Clutch

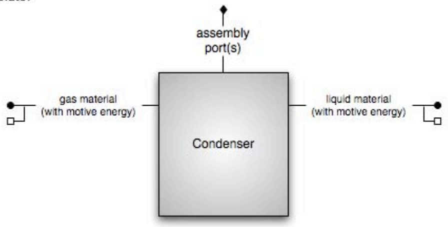
Function Template:



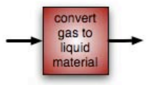
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Condenser

Port Template:


assembly port(s)
gas material (with motive energy)
liquid material (with motive energy)
Condenser

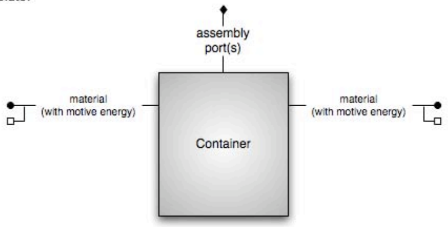
Function Template:



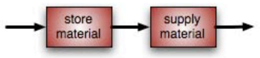
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Container

Port Template:



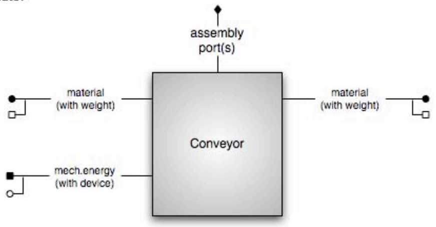
Function Template:



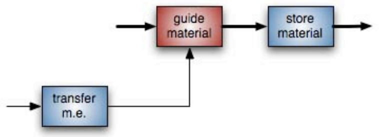
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Conveyor

Port Template:



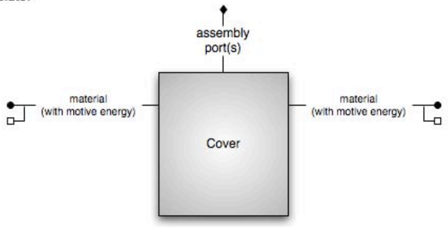
Function Template:



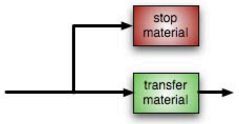
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Cover

Port Template:



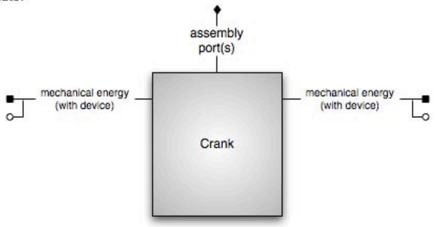
Function Template:



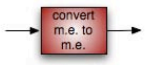
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Crank

Port Template:


assembly port(s)
mechanical energy (with device)
Crank
mechanical energy (with device)

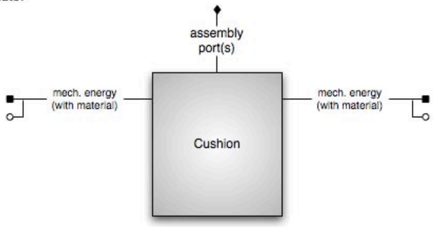
Function Template:



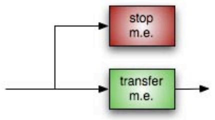
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Cushion

Port Template:



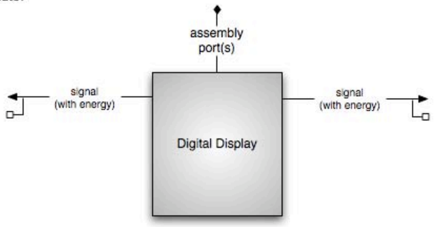
Function Template:



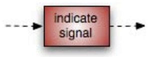
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Digital Display

Port Template:



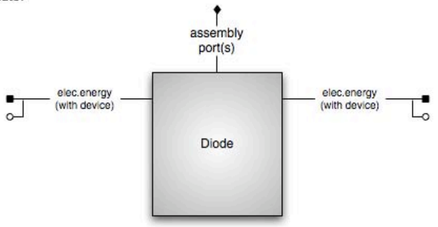
Function Template:



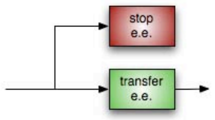
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Diode

Port Template:



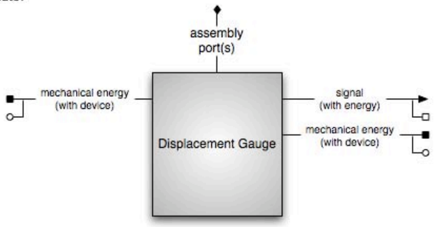
Function Template:



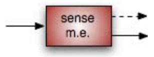
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Displacement Gauge

Port Template:



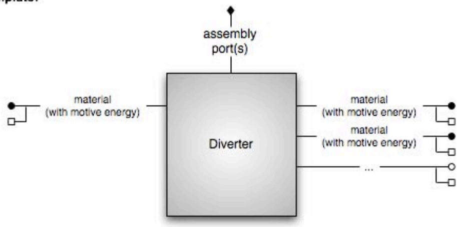
Function Template:



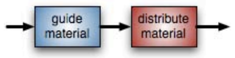
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Diverter

Port Template:



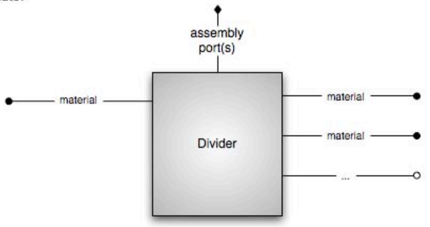
Function Template:



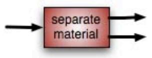
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Divider

Port Template:



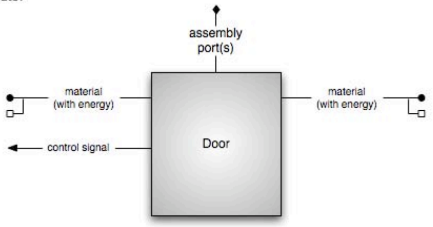
Function Template:



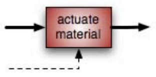
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Door

Port Template:



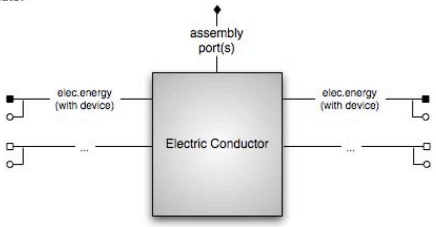
Function Template:



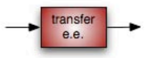
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Electric Conductor

Port Template:



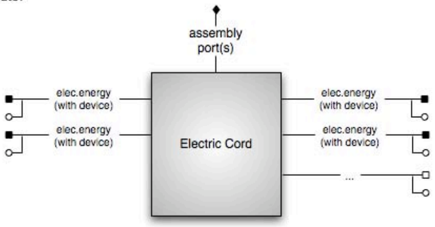
Function Template:



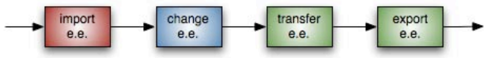
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Electric Cord

Port Template:


assembly port(s)
elec.energy (with device)
elec.energy (with device)
Electric Cord
elec.energy (with device)
elec.energy (with device)
...

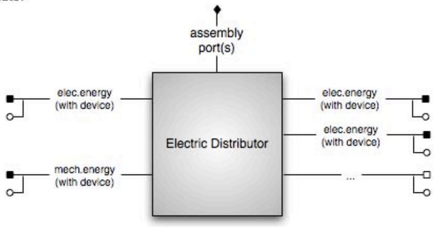
Function Template:



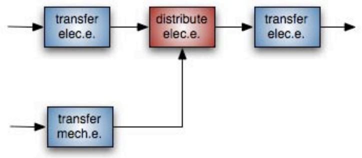
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Electric Distributor

Port Template:


assembly port(s)
elec.energy (with device)
mech.energy (with device)
Electric Distributor
elec.energy (with device)
elec.energy (with device)
...

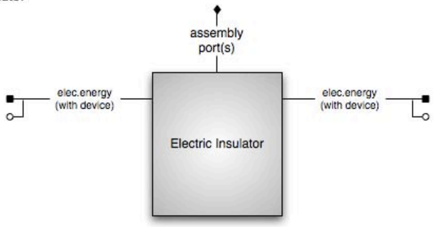
Function Template:



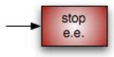
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Electric Insulator

Port Template:



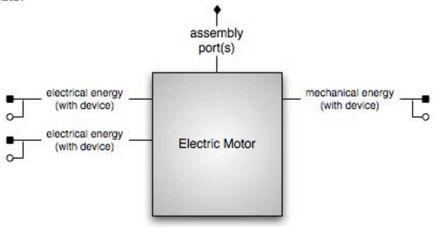
Function Template:



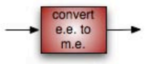
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Electric Motor

Port Template:


assembly port(s)
electrical energy (with device)
electrical energy (with device)
Electric Motor
mechanical energy (with device)

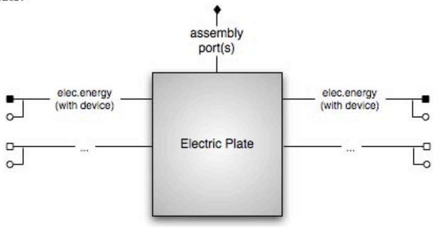
Function Template:



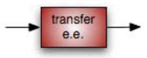
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Electric Plate *(Subset of Electric Conductor)*

Port Template:



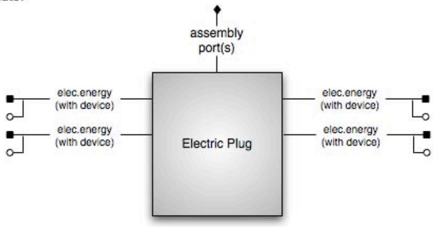
Function Template:



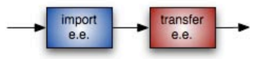
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Electric Plug

Port Template:



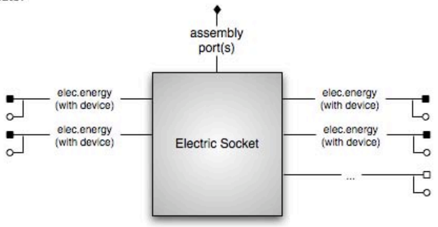
Function Template:



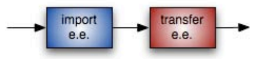
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Electric Socket

Port Template:



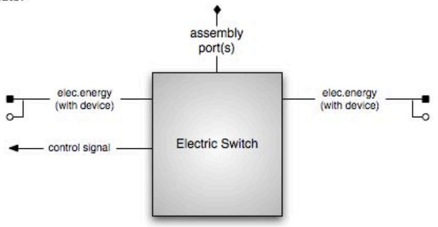
Function Template:



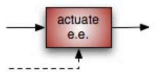
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Electric Switch

Port Template:



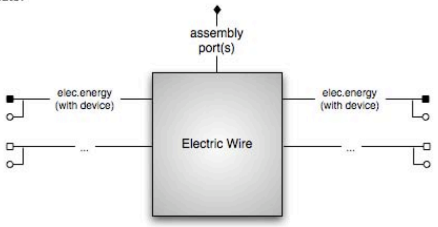
Function Template:



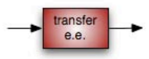
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Electric Wire *(Subset of Electric Conductor)*

Port Template:



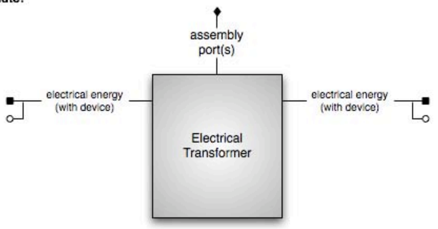
Function Template:



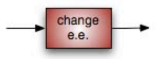
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Electrical Transformer

Port Template:


assembly port(s)
electrical energy (with device)
Electrical Transformer
electrical energy (with device)

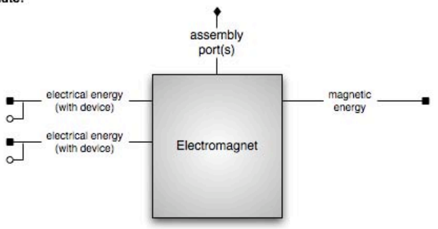
Function Template:



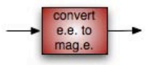
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Electromagnet

Port Template:


assembly port(s)
electrical energy (with device)
electrical energy (with device)
Electromagnet
magnetic energy

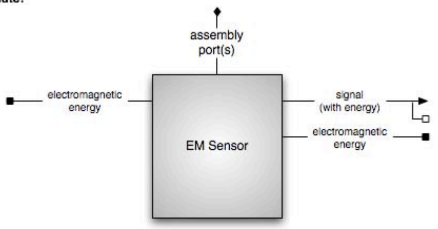
Function Template:



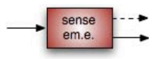
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: EM Sensor

Port Template:



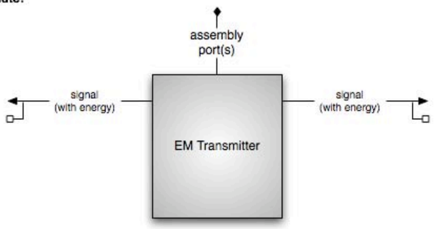
Function Template:



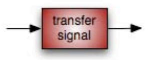
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: EM Transmitter

Port Template:



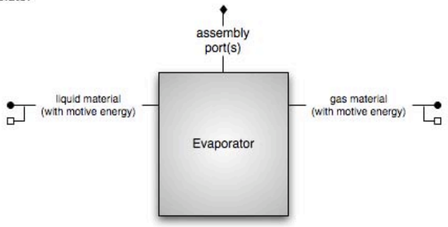
Function Template:



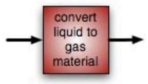
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Evaporator

Port Template:


assembly port(s)
liquid material (with motive energy)
Evaporator
gas material (with motive energy)

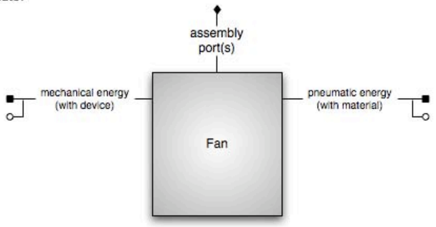
Function Template:



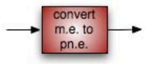
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Fan

Port Template:


assembly port(s)
mechanical energy (with device)
Fan
pneumatic energy (with material)

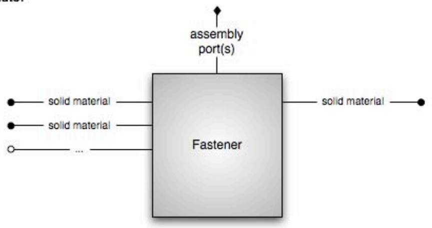
Function Template:



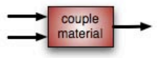
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Fastener

Port Template:



Function Template:

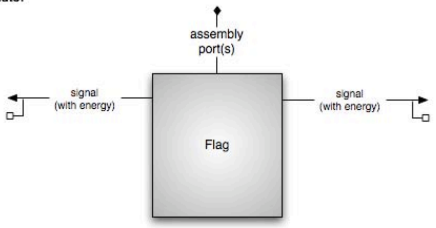


*****Only supporting functionality*****

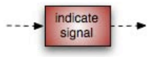
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Flag

Port Template:



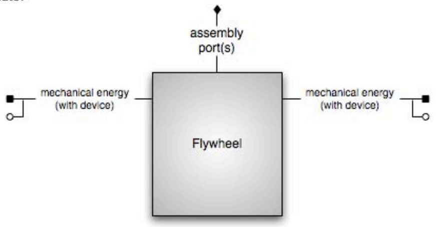
Function Template:



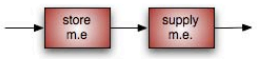
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Flywheel

Port Template:


assembly port(s)
mechanical energy (with device)
Flywheel
mechanical energy (with device)

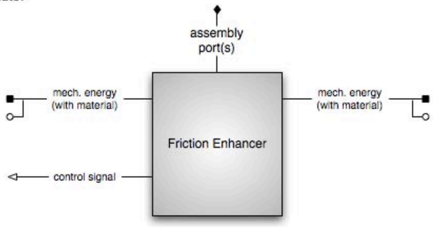
Function Template:



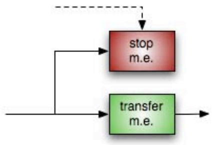
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Friction Enhancer

Port Template:



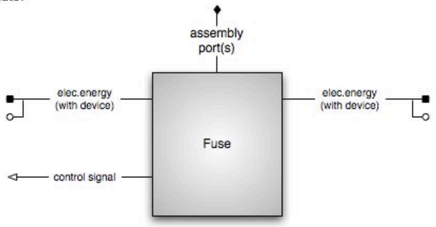
Function Template:



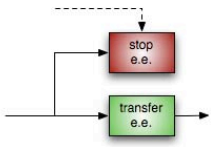
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Fuse

Port Template:



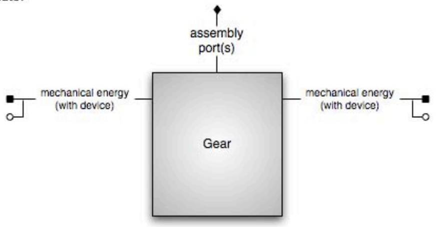
Function Template:



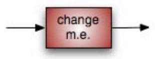
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Gear

Port Template:


assembly port(s)
mechanical energy (with device)
Gear
mechanical energy (with device)

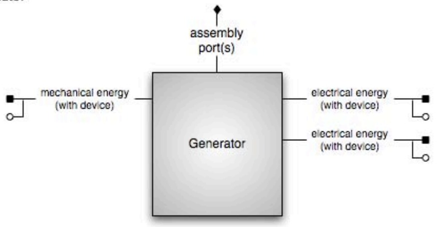
Function Template:



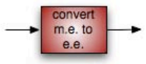
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Generator

Port Template:


assembly port(s)
mechanical energy (with device)
electrical energy (with device)
electrical energy (with device)
Generator

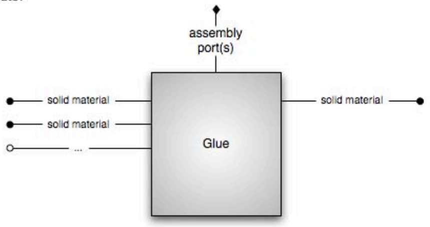
Function Template:



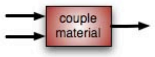
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Glue

Port Template:



Function Template:

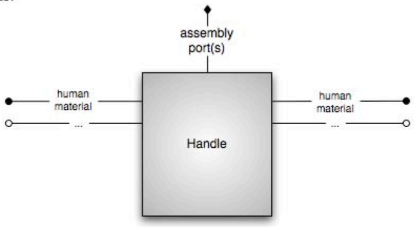


*****Only supporting functionality*****

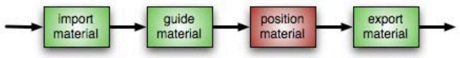
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Washer

Port Template:



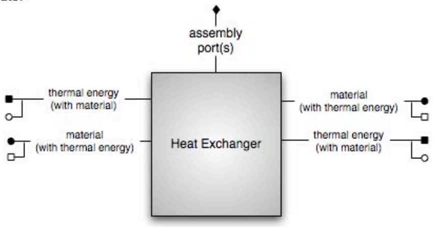
Function Template:



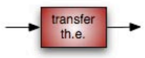
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Heat Exchanger

Port Template:


assembly port(s)
thermal energy (with material)
material (with thermal energy)
Heat Exchanger
material (with thermal energy)
thermal energy (with material)

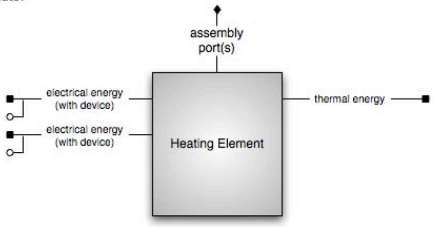
Function Template:



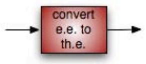
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Heating Element

Port Template:



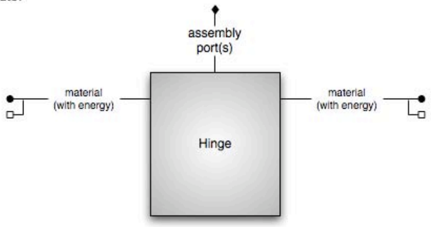
Function Template:



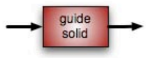
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Hinge

Port Template:



Function Template:

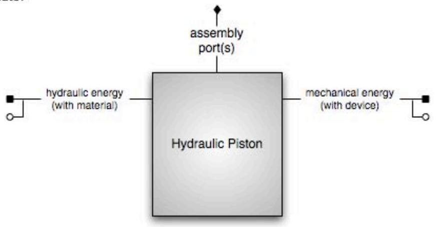


*****Only supporting functions*****

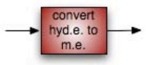
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Hydraulic Piston

Port Template:



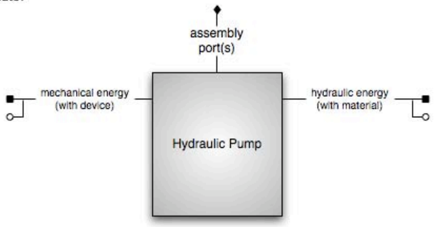
Function Template:



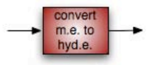
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Hydraulic Pump

Port Template:



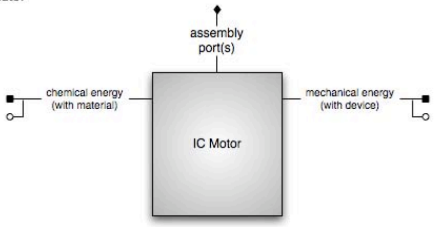
Function Template:



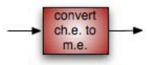
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: IC Motor

Port Template:



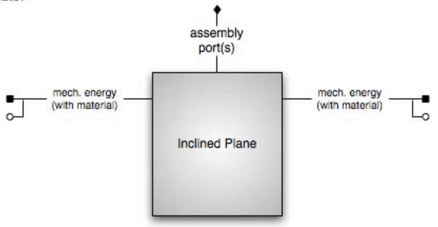
Function Template:



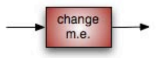
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Inclined Plane

Port Template:



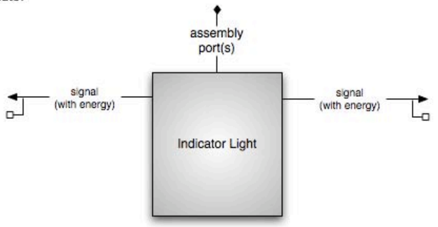
Function Template:



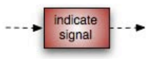
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Indicator Light

Port Template:



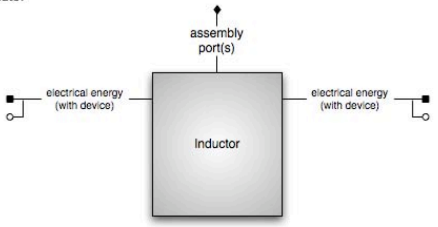
Function Template:



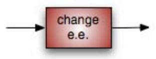
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Inductor

Port Template:



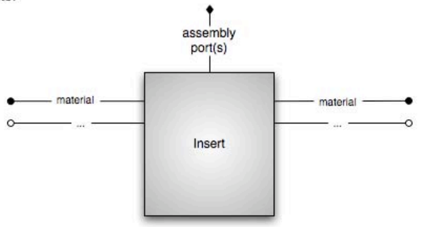
Function Template:



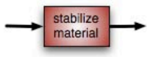
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Insert

Port Template:



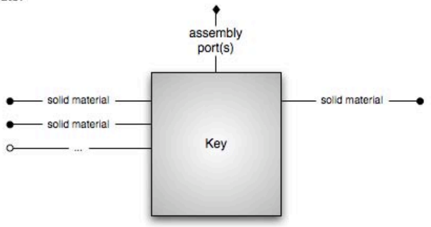
Function Template:



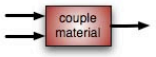
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Key

Port Template:



Function Template:

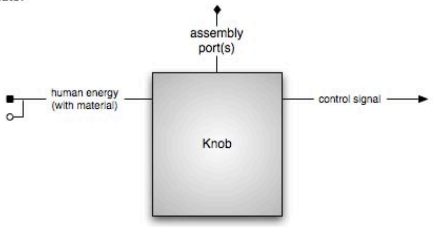


*****Only supporting functionality*****

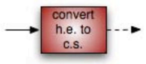
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Knob

Port Template:



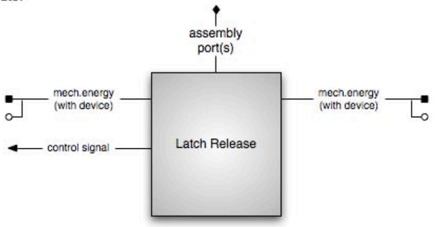
Function Template:



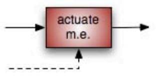
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Latch Release

Port Template:



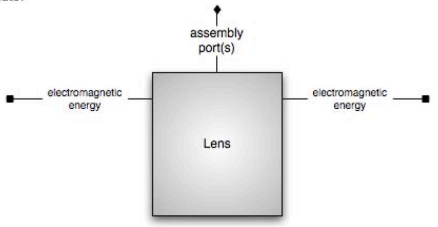
Function Template:



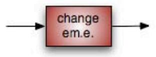
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Lens

Port Template:



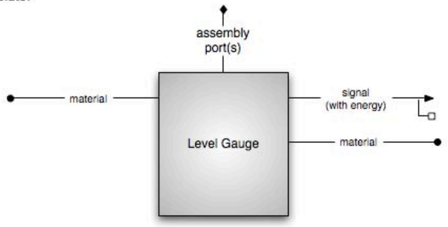
Function Template:



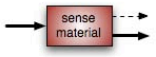
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Level Gauge

Port Template:



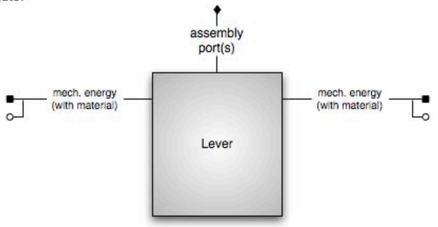
Function Template:



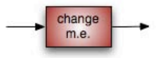
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Lever

Port Template:



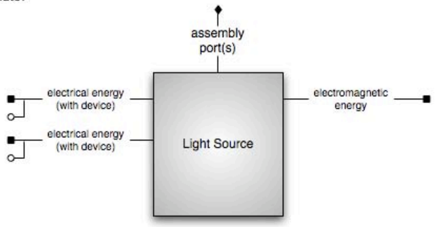
Function Template:



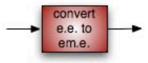
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Light Source

Port Template:



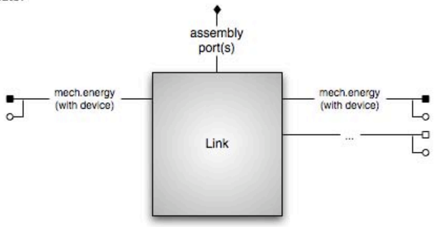
Function Template:



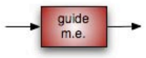
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Link

Port Template:



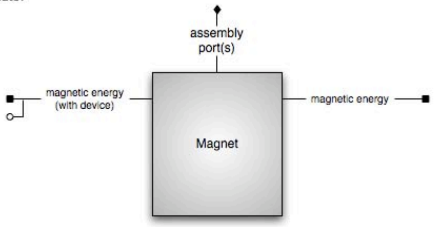
Function Template:



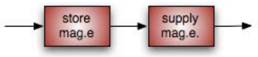
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Magnet

Port Template:



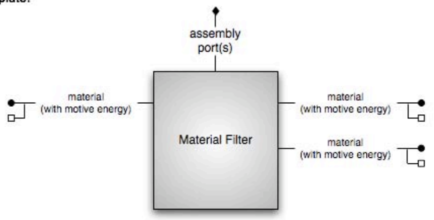
Function Template:



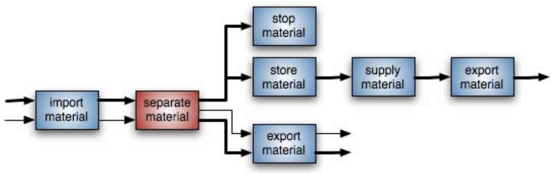
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Material Filter

Port Template:


assembly port(s)
material (with motive energy)
Material Filter
material (with motive energy)
material (with motive energy)

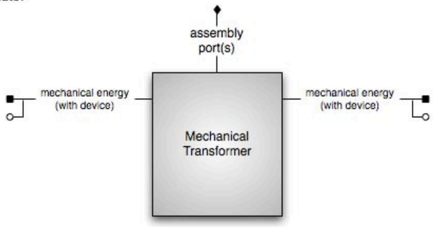
Function Template:



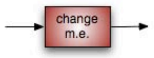
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Mechanical Transformer

Port Template:


assembly port(s)
mechanical energy (with device)
Mechanical Transformer
mechanical energy (with device)

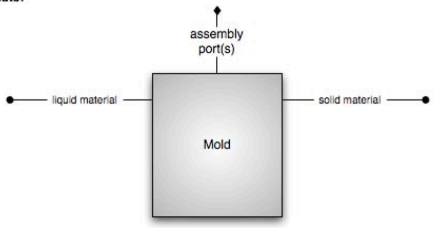
Function Template:



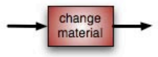
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Mold

Port Template:



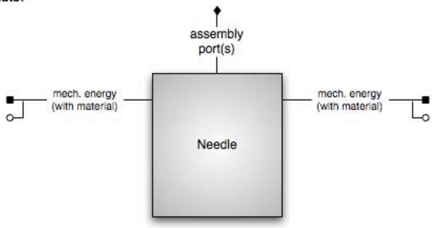
Function Template:



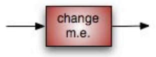
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Needle

Port Template:



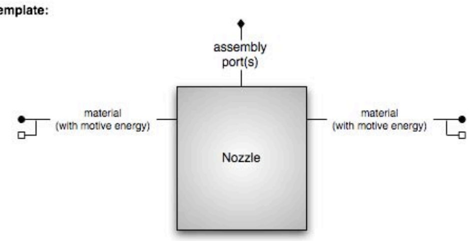
Function Template:



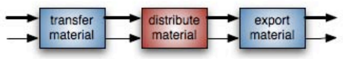
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Nozzle

Port Template:



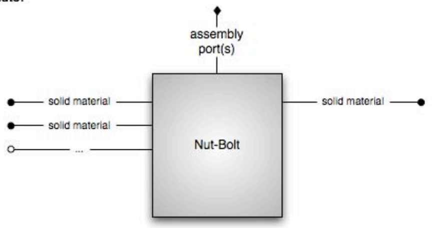
Function Template:



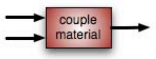
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Nut-Bolt

Port Template:



Function Template:

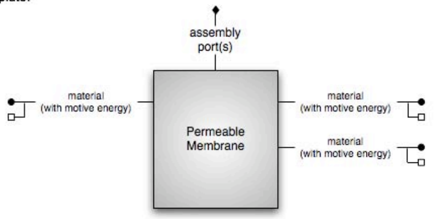


Only supporting functionality

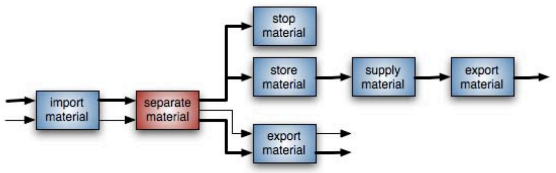
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Permeable Membrane *(Subset of Material Filter)*

Port Template:


assembly port(s)
material (with motive energy)
Permeable Membrane
material (with motive energy)
material (with motive energy)

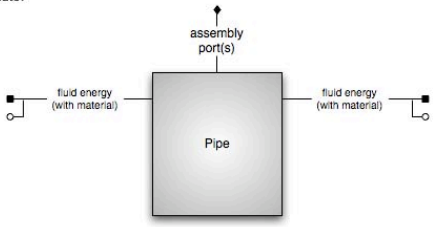
Function Template:



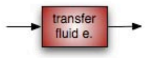
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Pipe

Port Template:



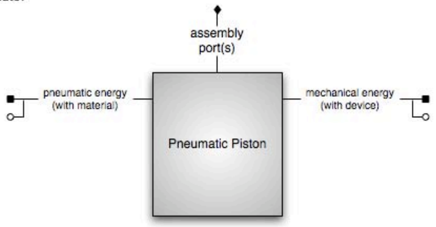
Function Template:



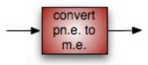
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Pneumatic Piston

Port Template:


assembly port(s)
pneumatic energy (with material)
Pneumatic Piston
mechanical energy (with device)

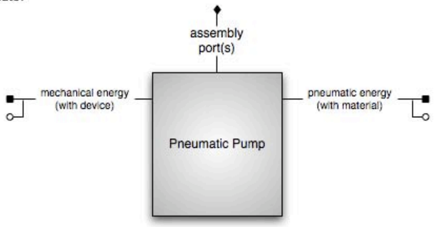
Function Template:



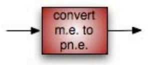
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Pneumatic Pump

Port Template:


assembly port(s)
mechanical energy (with device)
Pneumatic Pump
pneumatic energy (with material)

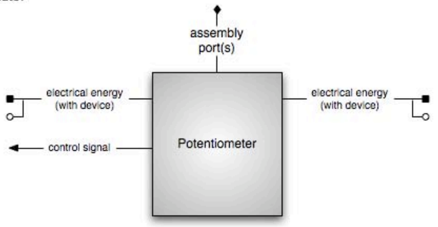
Function Template:



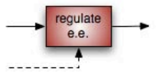
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Potentiometer

Port Template:


assembly port(s)
electrical energy (with device)
control signal
Potentiometer
electrical energy (with device)

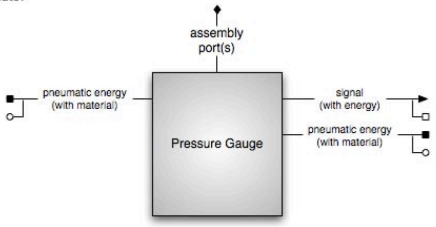
Function Template:



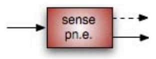
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Pressure Gauge

Port Template:


assembly port(s)
pneumatic energy (with material)
Pressure Gauge
signal (with energy)
pneumatic energy (with material)

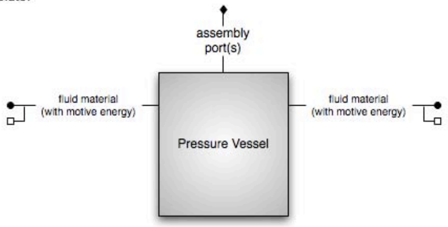
Function Template:



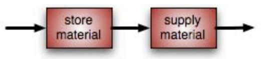
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Pressure Vessel

Port Template:


assembly port(s)
fluid material (with motive energy)
Pressure Vessel
fluid material (with motive energy)

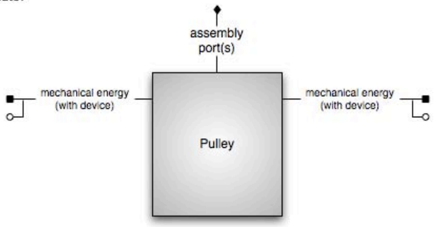
Function Template:



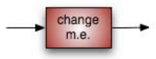
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Pulley

Port Template:


assembly port(s)
mechanical energy (with device)
Pulley
mechanical energy (with device)

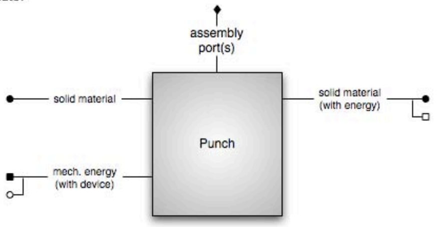
Function Template:



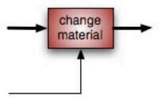
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Punch

Port Template:



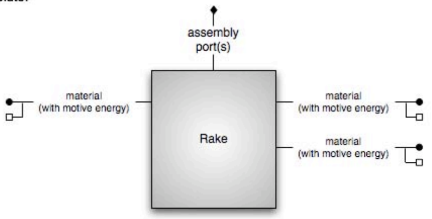
Function Template:



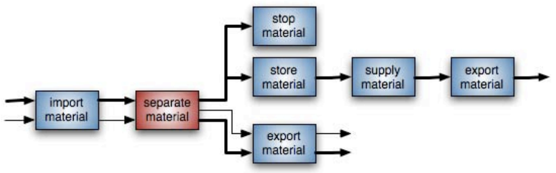
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Rake *(Subset of Material Filter)*

Port Template:


assembly port(s)
material (with motive energy)
Rake
material (with motive energy)
material (with motive energy)

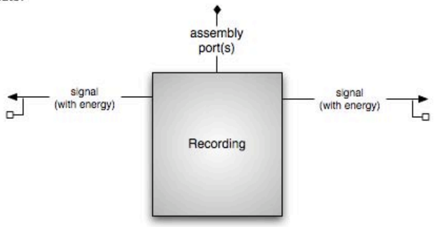
Function Template:



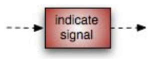
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Recording

Port Template:



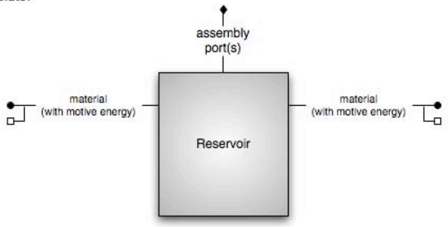
Function Template:



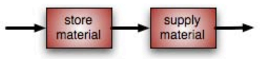
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Reservoir

Port Template:



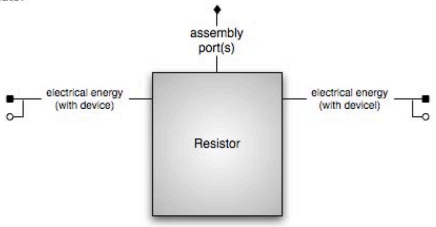
Function Template:



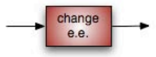
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Resistor

Port Template:


assembly port(s)
electrical energy (with device)
electrical energy (with device)
Resistor

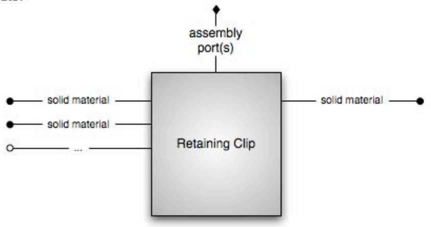
Function Template:



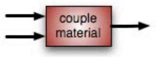
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Retaining Clip

Port Template:



Function Template:

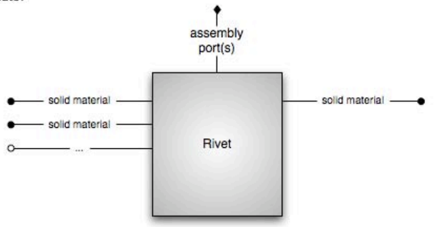


*****Only supporting functionality*****

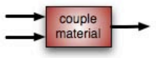
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Rivet

Port Template:



Function Template:

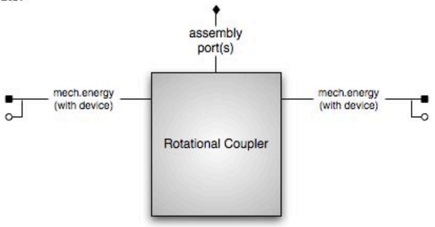


*****Only supporting functionality*****

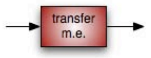
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Rotational Coupler

Port Template:



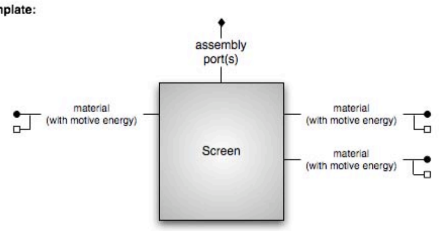
Function Template:



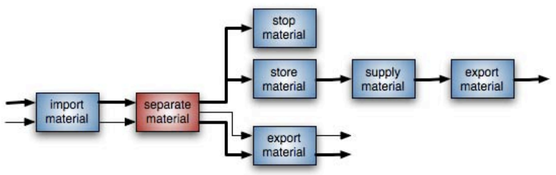
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Screen *(Subset of Material Filter)*

Port Template:


assembly port(s)
material (with motive energy)
Screen
material (with motive energy)
material (with motive energy)

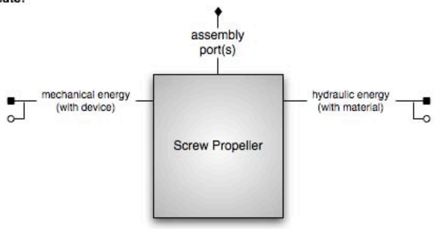
Function Template:



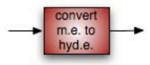
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Screw Propeller

Port Template:


assembly port(s)
mechanical energy (with device)
Screw Propeller
hydraulic energy (with material)

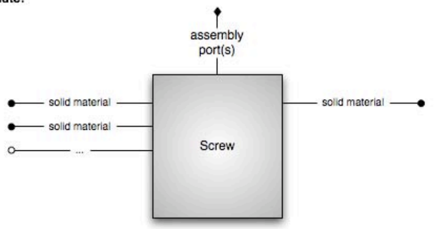
Function Template:



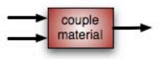
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Screw

Port Template:



Function Template:

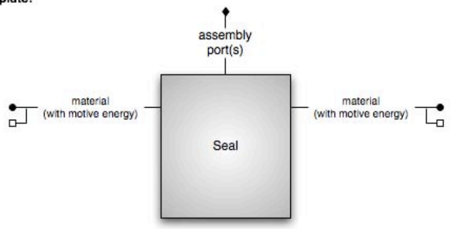


*****Only supporting functionality*****

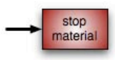
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Seal

Port Template:



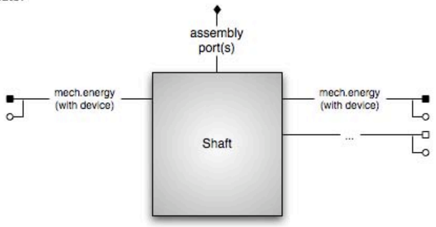
Function Template:



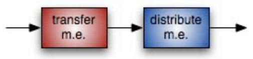
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Shaft

Port Template:



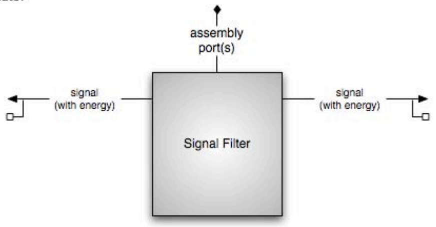
Function Template:



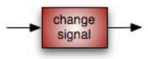
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Signal Filter

Port Template:



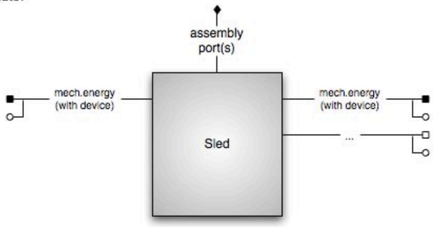
Function Template:



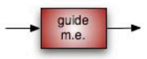
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Sled

Port Template:



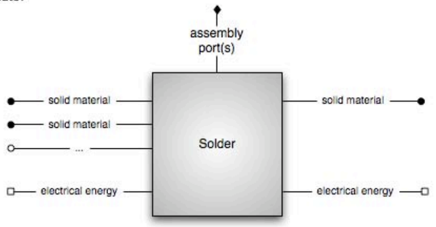
Function Template:



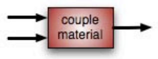
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Solder

Port Template:



Function Template:

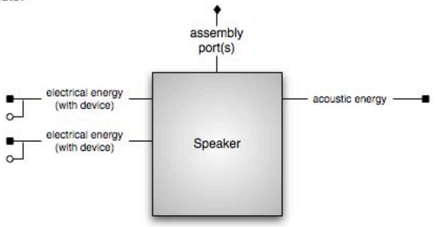


*****Only supporting functionality*****

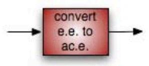
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Speaker

Port Template:


assembly port(s)
electrical energy (with device)
electrical energy (with device)
Speaker
acoustic energy

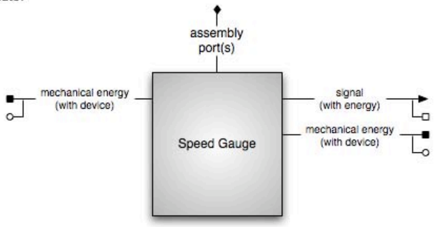
Function Template:



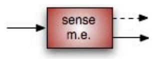
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Speed Gauge

Port Template:


assembly port(s)
mechanical energy (with device)
Speed Gauge
signal (with energy)
mechanical energy (with device)

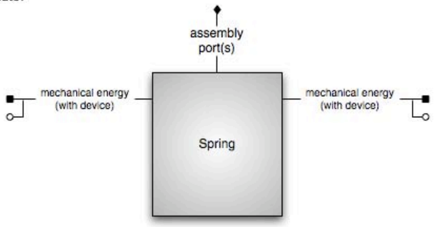
Function Template:



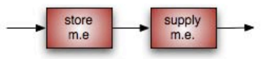
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Spring

Port Template:


assembly port(s)
mechanical energy (with device)
Spring
mechanical energy (with device)

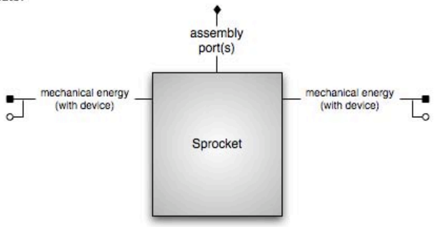
Function Template:



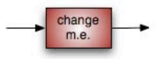
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Sprocket

Port Template:


assembly port(s)
mechanical energy (with device)
Sprocket
mechanical energy (with device)

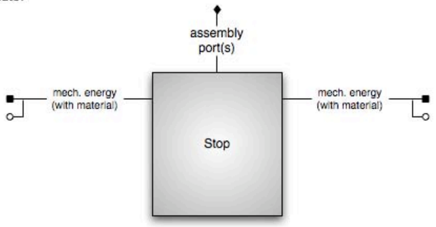
Function Template:



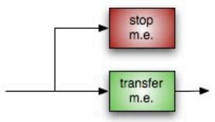
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Stop

Port Template:



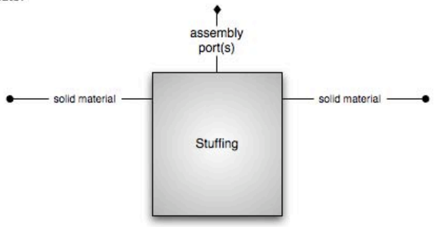
Function Template:



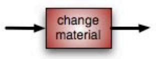
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Stuffing

Port Template:



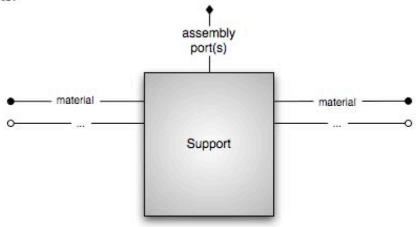
Function Template:



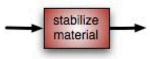
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Support

Port Template:



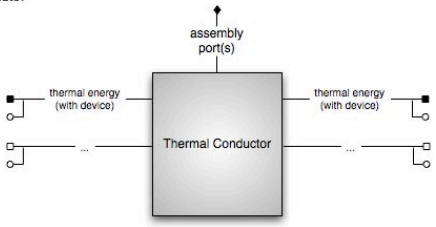
Function Template:



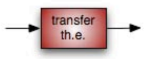
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Thermal Conductor

Port Template:



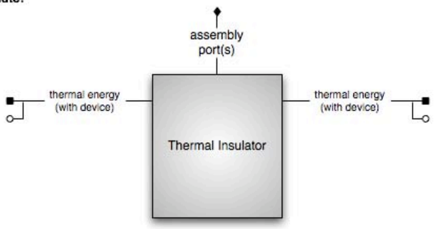
Function Template:



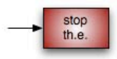
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Thermal Insulator

Port Template:



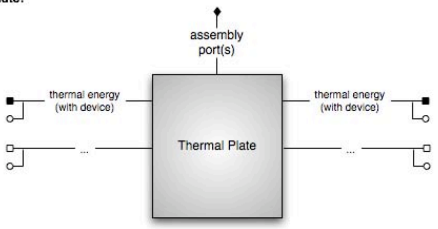
Function Template:



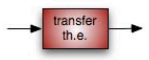
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Thermal Plate

Port Template:



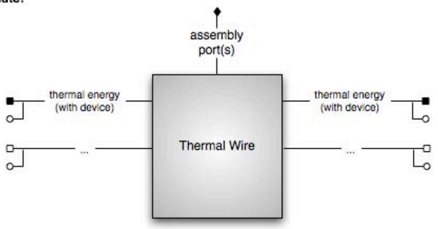
Function Template:



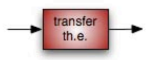
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Thermal Wire

Port Template:



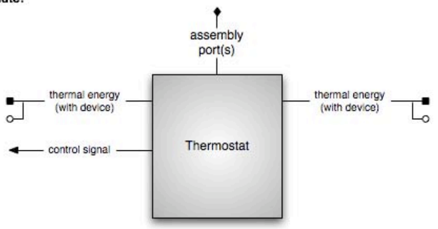
Function Template:



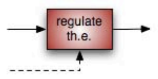
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Thermostat

Port Template:



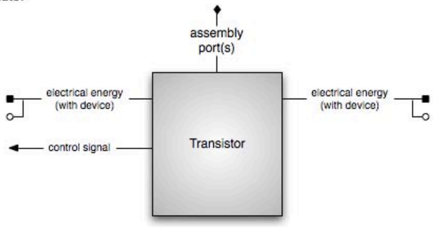
Function Template:



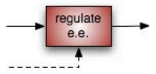
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Transistor

Port Template:


assembly port(s)
electrical energy (with device)
control signal
Transistor
electrical energy (with device)

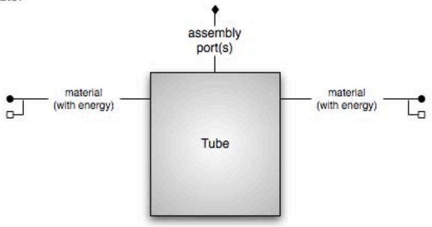
Function Template:



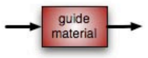
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Tube

Port Template:



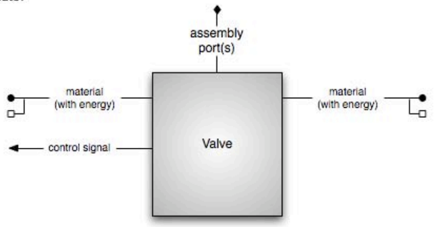
Function Template:



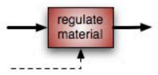
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Valve

Port Template:



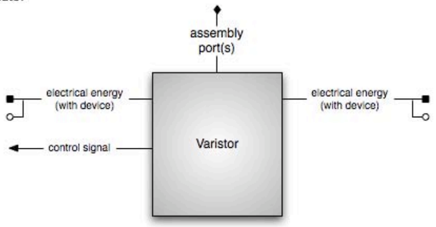
Function Template:



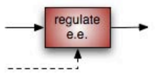
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Varistor

Port Template:


assembly port(s)
electrical energy (with device)
control signal
Varistor
electrical energy (with device)

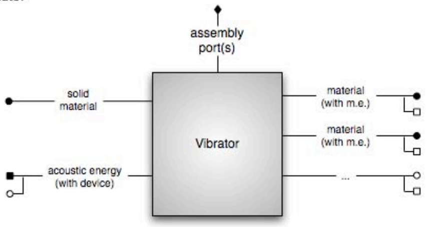
Function Template:



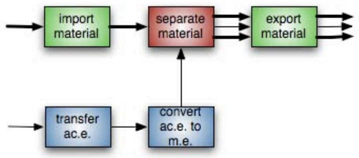
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Vibrator

Port Template:



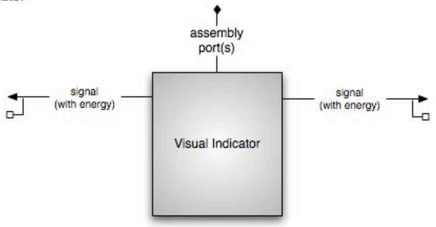
Function Template:



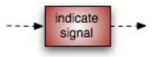
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Visual Indicator

Port Template:



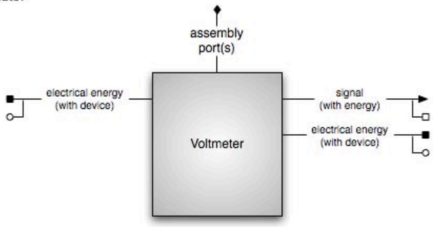
Function Template:



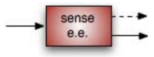
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Voltmeter

Port Template:


assembly port(s)
electrical energy (with device)
Voltmeter
signal (with energy)
electrical energy (with device)

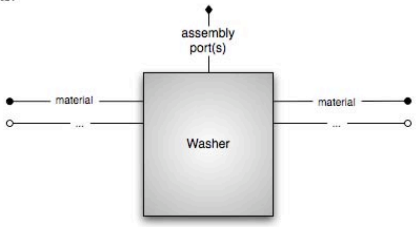
Function Template:



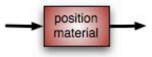
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Washer

Port Template:



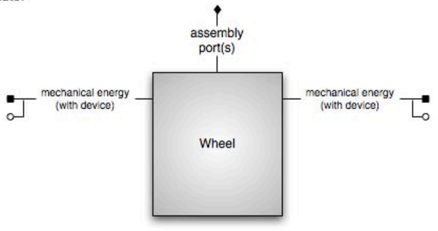
Function Template:



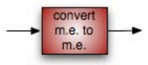
Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

Component Name: Wheel

Port Template:


assembly port(s)
mechanical energy (with device)
Wheel
mechanical energy (with device)

Function Template:



Key: 🟦 *Sometimes included* 🟩 *Usually included* 🟥 *Required (Black Box Function(s))*

APPENDIX C

Hierarchical Component Term List

Component Terms & Definitions				
Primary Component Classification	Secondary Component Classification	Component Term	Component Subset	Definition
Brancher				
	Separator			
		Abrasive		A device or material that uses texture on a surface to remove any portion of a firm (non-fluid) material.
		Blade		A device or material consisting of a broad flat or concave edge used to separate a firm (non-fluid) material.
		Centrifuge		A device that uses centrifugal force via a rapidly rotating container to separate fluid materials.
		Divider		A device that divides a material into smaller separate areas.
		Material Filter		A device or material consisting of a pattern of holes, slits, or pores used to separate constituents of a fluid mixture.
			Permeable Membrane	A material filter that uses a fine, porous, flexible material to separate particles from the surrounding mixture.
			Rake	A material filter that uses a series of parallel slits or tines to separate particles from the surrounding mixture.
			Screen	A material filter that uses a mesh structure to separate particles from the surrounding mixture.
		Scrub Brush		A device that uses bristles attached to a surface to remove any portion of a firm (non-fluid) material.
		Vibrator		A device that uses frequency oscillations to separate or dislodge a firm (non-fluid) material.
	Distributor			
		Brush		A device that uses bristles to distribute a fluid material over a surface.
		Diverter		A device or structure that distributes a flow of material into multiple directions by way of its geometry.
		Electric Distributor		A device used to systematically allocate electrical energy along multiple paths.
		Nozzle		A device at the end of a pipe, hose, or tube used to distribute a continuous flow of fluid material.
Channeler				
	Importer/Exporter			
		Electric Cord		A device used to bring electrical energy into a system from an external receptacle.
		Housing		A protective cover primarily used to bring flows into or out of a system that is also designed to contain or support components within it.
	Transferor			
		Belt		A device shaped as an endless loop of flexible material between two rotating shafts or pulleys used to transmit mechanical energy.
		Carousel		A device used to move material in a continuous circular path.
		Clutch		A device used to transmit rotational energy between two shafts that may be (dis)engaged smoothly.
		Conveyor		A device used to move material in a linear path.
		Electric Conductor		A device used to transmit electrical energy from one component to another.
			Electric Plate	An electric conductor in the form of a thin, flat sheet or strip.
			Electric Wire	An electric conductor in the form of a thin, flexible thread or rod.
		Electric Plug		A device in the form of a plug that transmits electrical energy via a detachable connection with an electric socket.
		Electric Socket		A device in the form of a receptacle that transmits electrical energy via a detachable connection with an electric plug.
		EM Transmitter		A device that transmits electromagnetic (EM) signals (such as infrared or RF) over a non-wired medium.

Component Terms & Definitions				
Primary Component Classification	Secondary Component Classification	Component Term	Component Subset	Definition
		Extender		A device that transmits mechanical energy between two elements of any jointed apparatus as they are drawn away from each other.
		Heat Exchanger		A device used to transmit heat from one medium to another.
		Pipe		A device in the forma of a hollow cylinder that transmits hydraulic or pneumatic energy by transferring fluid material under pressure.
		Projectile		A device that transmits mechanical energy by being thrown or propelled through the air.
		Rotational Coupler		A device used to connect coaxial shafts for power transmission from one to the other.
		Shaft		A device in the form of a cylindrical bar used to support rotating pieces or to transmit power or motion by rotation.
		Thermal Conductor		A device used to transmit thermal energy from one component to another.
			Thermal Plate	A thermal conductor in the form of a thin, flat sheet or strip.
			Thermal Wire	A thermal conductor in the form of a thin, flexible thread or rod.
	Guider			
		Bearing		A device in the form of a sphere or cylinder (or in an arrangement of spheres or cylinders) that is placed between moving parts to allow them to move easily relative to each other along a path.
		Hinge		A device that allows rigidly connected materials to rotate relative to each other about an axis, such as the revolution of a lid, valve, gate or door, etc.
		Link		A device connecting two or more components that transmits motive power from one part to another along a specific path.
		Sled		A device either under or within a machine used to facilitate the sliding of components relative to each another along a path.
		Tube		A device in the form of a hollow cylinder used to direct a fluid material (that is not under pressure) along a path.
Connector				
	Coupler			
		Clamp		A device used to hold two or more components together that is readily (dis)engageable without the use of an external tool.
		Fastener		A device used to hold two or more components together indefinitely with great effort or an external tool required to separate the joined components.
			Glue	A fastener in the form of an adhesive substance.
			Key	A fastener in the form of a piece of material that is inserted between other pieces, usually a pin-, bolt-, or wedge-like artifact fitting into a hole or space.
			Nut-Bolt	A fastener in the form of a threaded pin that screws into a usually square or hexagonal material through a threaded hole.
			Retaining Clip	A fastener in the form of a brace, band, or clasp.
			Rivet	A fastener in the form of a heavy pin having a head at one end with the other end hammered flat after being passed through holes in the joined pieces.
			Screw	A fastener in the form of a threaded pin, which does not require a nut to remain secure.
			Solder	A fastener in the form of a low-melting alloy used to join less fusible metals.
	Mixer			
		Agitator		A device used to maintain fluidity and plasticity, and to prevent segregation of liquids and solids in liquids, such as concrete and mortar.
		Carburetor		A device used to mix air with a fine spray of liquid fuel.
Magnitude Controller				
	Actuator			
		Door		A device in the form of a movable barrier, usually turning on hinges or sliding in a groove, and serving to close or open a passage into a space.

Component Terms & Definitions				
Primary Component Classification	Secondary Component Classification	Component Term	Component Subset	Definition
		Electric Switch		A device for making or breaking the flow of electrical energy in an electrical circuit.
		Latch Release		A device that is designed to hold or free a mechanism as required.
	Regulator			
		Potentiometer		A device used to adjust the flow of electrical energy in an electric circuit.
		Thermostat		A device used to adjust temperature by starting or stopping the supply of heat.
		Transistor		A semiconductor device with three connections capable of regulating the flow of electrical energy in an electrical circuit.
		Valve		A device by which the flow of a fluid material may be adjusted by opening, shutting, or partially obstructing one or more ports or passageways.
		Varistor		A device used to adjust the flow of electrical energy in an electric circuit.
	Changer			
		Capacitor		A device used to alter a signal by storing an electrical charge.
		Choke		A device in the form of a restriction in a pipe that reduces the flow of a fluid material.
		Electrical Transformer		A device used to change the voltage of an alternating electric current via a magnetic coupling between two separate circuits.
		Inclined Plane		A device in the form of a surface sloped at an angle to a reference surface, which provides a mechanical advantage for raising loads.
		Inductor		A device used to alter a signal by storing energy as a magnetic field.
		Lens		A device in the form of a translucent substance used to alter the path of optical energy transmitted through it.
		Lever		A device fixed at a fulcrum and acted on at two other points by two forces, each tending to cause it to rotate in opposite directions round the fulcrum.
		Mechanical Transformer		A device that alters the flow of mechanical energy during the process of transmitting force and motion between rotating or translating components.
			Gear	A mechanical transformer in the form of a disc or plate that transmits mechanical energy to another device by means of teeth.
			Pulley	A mechanical transformer in the form of a wheel or drum fixed on a shaft and turned by a belt, chain, or strap.
			Sprocket	A mechanical transformer in the form of a toothed wheel that engages a power chain.
		Mold		A hollow device used to give shape to a molten or hot fluid when it cools and hardens.
		Needle		A device in the form of a slender, usually pointed, rod used to amplify a mechanical rotation on a dial or other measuring instrument.
		Punch		A device used to make holes, impress a design, or stamp a die into a firm material.
		Resistor		A device that alters the flow of electrical energy by resisting the passage of electrical current.
		Signal Filter		A device that alters the frequency spectrum of signals passing through it.
		Stuffing		A device used to fill up hollows and to fill out or expand the outlines of the body.
	Stoppers			
		Acoustic Insulator		A device used to prevent the passage of sound, or vibration.
		Cap		A device in the form of a firm material secured to and used to prevent the flow of material into a hole or aperture.
		Check Valve		A device that allows a fluid to flow in only one direction.

Component Terms & Definitions				
Primary Component Classification	Secondary Component Classification	Component Term	Component Subset	Definition
		Cover		A device that overspreads an object, which is used to hide, defend, or shelter a material.
		Cushion		A device in the form of a soft pad or bumper used to prevent the transmission of mechanical energy from jarring, friction, or pressure.
		Diode		A semiconductor device which allows current to flow in only one direction.
		Electric Insulator		A device used to prevent the passage of electrical energy.
		Friction Enhancer		A device in the form of a material used to reduce heat and increase friction.
		Fuse		A device that breaks the flow of electrical energy in an electrical circuit in response to an excessive current.
		Seal		A device used prevent the flow of a fluid material, especially at a place where two surfaces meet.
		Stop		A device in the form of a rigid structure that is automatically activated by a predetermined displacement to limit the operation of a system.
		Thermal Insulator		A device used to prevent the passage of thermal energy.
Converter				
	Material Converter			
		Catalytic Converter		A device used to chemically transform a harmful gas material into one or more inert forms.
		Condenser		A device used to transform a gas material into a liquid material.
		Evaporator		A device used to transform a liquid material into a gas material.
	Energy Converter			
		Airfoil		A device with curved surfaces used to transform pneumatic energy into translational energy.
		Armature		A device used to transform magnetic energy into rotational energy.
		Burner		A device used to transform chemical energy into thermal energy.
		Cam		A device in the form of an eccentric curved wheel or disc used to transform rotational energy into reciprocating translational energy.
		Crank		A device used to transform reciprocating translational energy into rotational energy.
		Electric Motor		A device used to transform electrical energy into mechanical energy.
		Electromagnet		A device used to transform electrical energy into magnetic energy.
		Fan		A device in the form of a rotating shaft with two or more broad, angled blades attached used to transform rotational energy into pneumatic energy.
		Generator		A device used to transform mechanical energy into electrical energy.
		Heating Element		A device used to transform electrical energy into thermal energy.
		Hydraulic Piston		A device in the form of a cylinder tightly fitted inside a tube used to transform hydraulic energy into translational energy.
		Hydraulic Pump		A device used to transform mechanical energy into hydraulic energy by altering the pressures within a system.
		IC Motor		A device used to transform chemical energy in the form of liquid fuel into mechanical energy.
		Light Source		A device used to transform electrical energy into the spectrum of electromagnetic energy visible to humans.
		Pneumatic Piston		A device in the form of a cylinder tightly fitted inside a tube used to transform pneumatic energy into translational energy.

Component Terms & Definitions				
Primary Component Classification	Secondary Component Classification	Component Term	Component Subset	Definition
		Pneumatic Pump		A device used to transform mechanical energy into pneumatic energy by altering the pressures within a system.
		Screw Propeller		A device in the form of a rotating shaft with two or more broad, angled blades attached used to transform rotational energy into hydraulic energy.
		Speaker		A device used to transform an electrical signal into acoustic energy.
		Wheel		A device in the form of a disc or circle used to transform translational energy applied at the hub into rotational energy.
	Signal Converter			
		Knob		A device used to transform human energy into a control signal.
Provisioner				
	Material Supplier			
		Bladder		A device in the form of a hollow, expandable sac or membrane with a narrow opening used to accumulate and dispense a material.
		Container		A device in the form of a closed canister used to accumulate and dispense a material.
		Pressure Vessel		A device in the form of a sealed tank used to accumulate and dispense a pressurized fluid material.
		Reservoir		A device in the form of an open tank used to accumulate and dispense a material.
	Energy Supplier			
		Battery		A device used to accumulate and dispense electrical energy by means of a chemical reaction.
		Flywheel		A device used to accumulate and dispense rotational energy via angular momentum.
		Spring		A device used to accumulate and dispense mechanical energy via the elastic properties of the device's material properties.
	Signal Supplier			
Signaler				
	Sensor			
		Ammeter		A device used to determine the current through an electric circuit.
		Displacement Gauge		A device used to determine translational or rotational distance in a system.
		EM Sensor		A device used to detect an electromagnetic signal.
		Level Gauge		A device in the form of an external plate or face on which the amount of a fluid material is determined.
		Pressure Gauge		A device used to determine the pressure from hydraulic or pneumatic energy in a system.
		Speed Gauge		A device used to determine velocity in a system.
		Voltmeter		A device used to determine the voltage across a portion of an electric circuit.
	Indicator			
		Visual Indicator		A device used to visibly indicate a signal.
			Analog Display	A visual indicator in the form of a continuously variable dial or gauge.
			Digital Display	A visual indicator in the form of a discrete readout or gauge.
			Flag	A visual indicator in the form of a physical banner or marker.
			Indicator Light	A visual indicator in the form of a single bulb.
		Auditory Indicator		A device used to acoustically indicate a signal.

Component Terms & Definitions				
Primary Component Classification	Secondary Component Classification	Component Term	Component Subset	Definition
			Bell	An auditory indicator in the form of a hollow object that is struck to produce vibration.
			Buzzer	An auditory indicator in the form of an electronic device that emits a buzzing noise.
			Recording	An auditory indicator in the form of stored acoustic information that is replayed.
	Processor			
		Circuit Board		A device in the form of a printed circuit used to perform systematic operations on a signal.
Supporter				
	Stabilizer			
		Insert		A device in the form of a material around which another material sets, solidifies, or is formed and used to strengthen or prevent a material from overturning.
		Support		A device that holds up or sustains the weight of a body.
	Securer			
		Bracket		A device in the form of a piece or combination of pieces, usually triangular in general shape, projecting from, or fastened to, a wall, or other surface, to secure heavy bodies or angles.
	Positioner			
		Handle		A device used to place a human hand in an appropriate configuration for grasping or interacting.
		Washer		A device in the form of a disk or ring used to provide spacing between components located on a axle or shaft.

APPENDIX D

Analytical Design Structure Matrix

	Abrasive	Acoustic Insulator	Agitator	Airfoil	Ammeter	Analog Display	Armature	Battery	Bearing	Bell	Belt	Bladder	Blade	Brush	Burner	Buzzer	Cam	Cap	Capacitor	Carburetor	Carousel	Catalytic Converter	Centrifuge	Check Valve	Choke	Circuit Board	Clamp	Clutch	Condenser	Container	Conveyor	Cover	Crank	Cushion	Digital Display	Diode	Displacement Gauge	Diverter	Divider	Door	Electric Cord	Electric Distributor	Electric Insulator	Electric Motor	Electric Plate	Electric Plug	Electric Socket	Electric Switch	Electric Wire	Electrical Transformer	Electromagnet	EM Sensor	EM Transmitter	Evaporator	Fan	Flag	Flywheel	Friction Enhancer	Fuse	Gear	Generator	Glue	Handle	Heat Exchanger	Heating Element	Hinge	Housing	Hydraulic Piston	Hydraulic Pump	IC Motor	Inclined Plane	Indicator Light	Inductor	Insert	Key	Knob	Latch Release	Lens	Level Gauge	Lever	Light Source	Link	Mold
Abrasive	1	0	0	0	0	1	0	0	1	1	1	1	1	1	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	0	1	0	0	0	1	1	0	1	0	0	0	0	0	0	1	0	0	1	0	1	0
Acoustic Insulator	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Agitator	0	0	1	0	0	1	0	0	0	1	0	1	1	1	0	0	0	0	0	1	1	1	1	0	0	0	1	0	1	1	1	1	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	1	1	1	0	1	1	0	0	0	0	0	0	1	1	0	0	0	1	0	0	0	1
Airfoil	0	0	0	1	0	1	0	0	1	1	1	0	1	0	0	0	1	0	0	0	1	0	1	1	0	0	0	1	0	0	1	0	1	1	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	1	1	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	1	0	0	1	0	1	0
Ammeter	0	0	0	0	1	1	0	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	0	1	0	0	0	0	0	1	0	1	0	0	0	1	0	1	0	0	1	0	1	1	0	0	0	0	0	0	0	1	0	0
Analog Display	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1
Armature	0	0	0	0	0	1	1	0	1	1	1	0	1	0	0	0	1	0	0	0	1	0	1	0	0	0	0	1	0	0	1	0	1	1	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	1	1	1	1	0	1	1	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0	1	0	0	1	0	1	0
Battery	0	0	0	0	1	1	0	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	0	1	0	0	0	0	0	1	0	1	0	0	0	1	0	1	0	0	1	0	1	1	0	0	0	0	0	0	0	1	0	0
Bearing	1	0	0	1	0	1	1	0	1	1	1	0	1	0	0	0	1	0	0	0	1	0	1	0	0	0	0	1	0	0	1	0	1	1	0	0	1	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	1	1	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	1	1	0	1	0
Bell	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1
Belt	1	0	0	1	0	1	1	0	1	1	1	0	1	0	0	0	1	0	0	0	1	0	1	0	0	0	0	1	0	0	1	0	1	1	0	0	1	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	1	1	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	1	1	0	1	0
Bladder	1	0	1	0	0	1	0	0	0	1	0	1	1	1	0	0	0	1	0	1	1	0	1	0	0	0	1	0	1	1	1	1	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1
Blade	1	0	1	1	0	1	1	0	1	1	1	1	1	1	0	0	0	1	0	0	1	0	1	0	0	0	1	0	1	1	1	1	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1	0	0	1	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1
Brush	1	0	1	0	0	1	0	0	0	1	0	1	1	1	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	1	1	0	0	1	1	1	0	1	0	0	0	1	1	0	1	0	0	1	0	1	0
Burner	0	0	0	0	0	1	0	0	0	1	0	0	0	0	1	0	0	1	0	1	0	1	0	1	1	0	0	0	1	1	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Buzzer	0	1	0	0	1	1	0	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	0	1	0	0	0	0	0	1	0	1	0	0	0	1	0	1	0	0	1	0	1	1	0	0	0	0	0	0	0	1	0	0
Cam	1	0	0	1	0	1	1	0	1	1	1	0	0	1	0	0	1	0	0	0	1	0	1	0	0	0	0	1	0	0	1	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	1	1	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	1	0	1	1	0	1	0
Cap	1	0	0	0	0	1	0	0	0	1	0	1	1	1	1	0	0	1	0	1	1	0	1	0	0	0	1	0	1	1	1	1	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Capacitor	0	0	0	0	1	1	0	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	0	1	0	0	0	0	0	1	0	1	0	0	0	1	0	1	0	0	1	0	1	1	0	0	0	0	0	0	0	1	0	0
Carburetor	0	0	1	0	0	1	0	0	0	1	0	1	0	0	1	0	0	1	0	1	0	1	0	1	0	0	0	0	1	0	0	1	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	1	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	1
Carousel	1	0	1	1	0	1	1	0	1	1	1	1	1	1	0	0	1	1	0	0	1	0	0	0	1	0	1	1	1	1	1	1	1	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	1	0	0	0	1	1	1	0	1	1	0	0	1	1	0	1	0	1	1	0	1	1
Catalytic Converter	0	0	1	0	0	1	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	1	0	1	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Centrifuge	0	0	1	1	0	1	1	0	1	1	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	0	0	1	1	1	0	1	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0	1	0	1	0	1	0	1	1	1	1	1	1	0	0	1	1	0	1	0	1	1	0	1	1
Check Valve	0	0	0	1	0	1	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	1	1	1	1	0	0	0	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Choke	0	0	0	0	0	1	0	0	0	1	0	0	0	0	1	0	0	0	0	0	1	0	1	1	1	0	0	0	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Circuit Board	0	0	0	0	1	1	0	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	0	1	0	0	0	0	0	1	0	1	0	0	0	1	0	1	0	0	1	0	1	1	0	0	0	0	0	0	0	1	0	0
Clamp	0	0	1	0	0	1	0	0	0	1	0	1	1	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0	0	1	1	1	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0	0	0	0	0	0	1	1	0	1	0	0	1	0	1	0
Clutch	0	0	0	1	0	1	1	0	1	1	1	0	0	0	0	0	1	0	0	0	1	0	1	0	0	0	0	1	0	0	1	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	1	1	0	0	0	0	0	1	1	1	1	1	0	0	0	0	1	1	0	1	1	0	1	0
Condenser	0	0	1	0	0	1	0	0	0	1	0	1	1	0	1	0	0	1	0	1	1	1	1	1	1	0	0	0	1	1	1	1	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Container	1	0	1	0	0	1	0	0	0	1	0	1	1	1	1	0	0	1	0	0	1	0	1	1	1	0	1	0	1	1	1	1	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	1	0	1	1	1	1	0	0	0	0	1	1	0	1	0	1	0	0	0	1
Conveyor	1	0	1	1	0	1	1	0	1	1	1	1	1	1	0	0	1	1	0	0	1	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	1	0	0	0	1	1	1	0	1	1	0	0	1	1	0	1	0	1	1	0	1	1
Cover	1	0	1	0	0	1	0	0	0	1	0	1	1	1	1	0	0	1	0	1	1	1	1	0	0	0	1	0	1	1	1	1	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Crank	0	0	0	1	0	1	1	0	1	1	1	0	0	0	0	0	1	0	0	0	1	0	1	0	0	0	0	1	0	0	1	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	1	1	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	1	0	1	1	0	1	0
Cushion	0	0	0	1	0	1	1	0	1	1	1	0	0	0	0	0	1	0	0	0	1	0	1	0	0	0	0	1	0	0	1	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	1	1	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	1	0	1	1	0	1	0
Digital Display	0	0	0	0	1	1	0	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	0	1	0	0	0	0	0	1	0	1	0	0	0	1	0	1	0	0	1	0	1	1	0	0	0	0	0	0	0	1	0	0
Diode	0	0	0	0	1	1	0	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	0	1	0	0	0	0	0	1	0	1	0	0	0	1	0	1	0	0	1	0	1	1	0	0	0	0	0	0	0	1	0	0
Displacement Gauge	0	0	0	1	0	1	1	0	1	1	1	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	0	0	1	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	1	1	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	1	0	1	1	0	1	0
Diverter	1	0	1	0	0	1	0	0	0	1	0	1	1	1	1	0	0	1	0	1	1	1	1	1	1	0	1	0	1	1	1	1	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Divider	1	0	1	0	0	1	0	0	0	1	0	1	1	1	0	0	0	1	0	1	1	0	0	0	0	0	1	0	1	1	1	1	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Door	1	0	1	0	0	1	0	0	0	1	0	1	1	1	1	0	0	1	0	0	1	0	0	0	0	0	1	0	1	1	1	1	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1
Electric Cord	0	0	0	0	1	1	0	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	0	1	0	0	0	0	0	1	0	1	0	0	0	1	0	1	0	0	1	0	1	1	0	0	0	0	0	0	0	1	0	0
Electric Distributor	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	0	1	0	0	0	0	0	1	0	1	0	0	0	1	0	1	0	0	1	0	1	1	0	0	0	0	0	0	0	1	0	0
Electric Insulator	0	0	0	0	1	1	0	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	0	1	0	0	0	0	0	1	0	1	0	0	0	1	0	1	0	0	1	0	1	1	0	0	0	0	0	0	0	1	0	0
Electric Motor	0	0	0	0	1	1	0	1	1	1	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	0	1	0	0	0	0	0	1	0	1	0	0	0	1	0	1	0	0	1	0	1	1	0	0	0	0	0	0	0	1	0	0
Electric Plate	0	0	0	0	1	1	0	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	0	1	0	0	0	0	0	1	0	1	0	0	0	1	0	1	0	0	1	0	1	1	0	0	0	0	0	0	0	1	0	0
Electric Plug	0	0	0	0	1	1	0	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	0	1	0	0	0	0	0	1	0	1	0	0	0	1	0	1	0	0	1	0	1	1	0	0	0	0	0	0	0	1	0	0
Electric Socket	0	0	0	0	1	1	0	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	0	1	0	0	0	0	0	1	0	1	0	0	0	1	0	1	0	0	1	0	1	1	0	0	0	0	0	0	0	1	0	0
Electric Switch	0	0	0	0	1	1	0	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	0	1	0	0	0	0	0	1	0	1	0	0	0	1	0	1	0	0	1	0	1	1	0	0	1	0	0	0	0	1	0	0
Electric Wire	0	0	0	0	1	1	0	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	0	1	0	0	0	0	0	1	0	1	0	0	0	1	0	1	0	0	1	0	1	1	0	0	0	0	0	0	0	1	0	0
Electrical Transformer	0	0	0	0	1	1	0	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	0	1	0	0	0	0	0	1	0	1	0	0	0	1	0	1	0	0	1	0	1	1	0	0	0	0	0	0	0	1	0	0
Electromagnet	0	0	0	0	1	1	1	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	0	1	0	0	0	0	0	1	0	1	0	0	0	1	0	1	0	0	1	0	1	1	0	0	0	0	0	0	0	1	0	0
EM Sensor	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1	0	1	0	0	0	1	0	1	0	0	1	0	1	1	0	0	0	0	0	0	0	1	0	0
EM Transmitter	0	0	0	0	1	1	0	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	1	0	1	0	0	0	1	0	1	0	0	1	0	1	1	0	0	0	0	0	0	0	1	0	0
Evaporator	0	0	1	0	0	1	0	0	0	1	0	1	1	0	1	0	0	1	0	1	0	1	1	1	1	0	0	0	1	1	0	1	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Fan	0	0	0	1	0	1	1	0	1	1	1	0	0	0	0	0	1	0	0	0	0	1	0	1	1	0	0	1	1	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	1	0	1	1	0	1	0	0	0	1	0	0	1	1	0	0	0	0	0	1	0	1	1	0	1	0
Flag	0	0	1	1	0	0	1	0	1	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1	1	0	1	0	0	0	1	0	1	1	1	0	0	0	0	0	1	0	1	1	0	1	0
Flywheel	1	0	0	1	0	1	1	0	1	1	1	0	0	1	0	0	1	0	0	0	1	0	1	0	0	0	0	1	0	0	1	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	1	1	0	1	0	0	0	1	1	1	1	1	0	0	0	0	0	1	0	1	1	0	1	0
Friction Enhancer	1	0	0	1	0	1	1	0	1	1	1	0	0	1	0	0	1	0	0	0	1	0	1	0	0	0	0	1	0	0	1	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	1	1	0	1	0	0	0	1	1	1	1	1	0	0	0	0	1	1	0	1	1	0	1	0
Fuse	0	0	0	0	1	1	0	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	0	1	0	0	0	0	0	1	0	1	0	0	0	1	0	1	0	0	1	0	1	1	0	0	1	0	0	0	0	1	0	0
Gear	1	0	0	1	0	1	1	0	1	1	1	0	0	1	0	0	1	0	0	0	1	0	1	0	0	0	0	1	0	0	1	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	1	1	0	1	0	0	0	1	1	1	1	1	0	0	0	0	0	1	0	1	1	0	1	0
Generator	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	1	1	0	1	0	0	0	0	0	0	1	0	1	0	0	0	0	1	1	1	1	1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	0	0	0	1	0	1	1	0	1	1	0	1	0	0	0	1	0	1	1	1	1	0
Glue	0	0	1	0	0	1	0	0	0	1	0	1	1	1	0	0	0	0	0	0	1	0	1	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
Handle	1	0	1	0	0	1	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	1	0	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	1	0	0	1	0	1	0
Heat Exchanger	0	0	1	0	0	1	0	0	0	1	0	0	0	0	1	0	0	1	0	1	0	0	1	1	1	0	0	0	1	1	0	1	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Heating Element	0	0	0	0	1	1	0	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	0	1	0	0	0	0	0	1	0	1	0	0	0	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0
Hinge	0	0	1	0	0	1	0	0	0	1	0	1	1	1	0	0	0	0	0	0	1	0	1	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
Housing	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0
Hydraulic Piston	1	0	0	0	0	1	0	0	1	1	1	0	0	1	0	0	1	0	0	1	1	0	1	1	1	0	0	1	1	1	1	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0	1	1	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0	1	1	0	1	0
Hydraulic Pump	0	0	0	0	0	1	1	0	1	1	1	0	0	0	1	0	1	0	0	1	0	0	1	1	1	0	0	1	1	1	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	0	1	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	1	0	1	1	0	1	0
IC Motor	1	0	0	0	1	1	0	1	1	1	1	0	0	1	0	1	1	0	1	0	1	0	1	0	0	1	0	1	0	0	1	0	1	1	1	1	1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	0	0	0	0	0	0	0	1	1	1	0	1	0	0	0	1	0	1	1	1	1	0
Inclined Plane	0	0	0	1	0	1	1	0	1	1	1	0	0	0	0	0	1	0	0	0	1	0	1	0	0	0	0	1	0	0	1	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	1	1	0	1	0	0	0	0	1	1	1	1	0	0	0	0	0	1	0	1	1	0	1	0
Indicator Light	0	0	0	0	1	1	0	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
Inductor	0	0	0	0	1	1	0	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	0	1	0	0	0	0	0	1	0	1	0	0	0	1	0	1	0	0	1	0	0	1	0	0	0	0	0	0	0	1	0	0
Insert	0	0	1	0	0	1	0	0	0	1	0	1	1	1	0	0	0	0	0	0	1	0	1	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
Key	0	0	1	0	0	1	0	0	0	1	0	1	1	1	0	0	0	0	0	0	1	0	1	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
Knob	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0
Latch Release	1	0	0	1	0	1	1	0	0	1	0	0	0	1	0	0	1	0	0	0	1	0	1	0	0	0	1	1	0	1	1	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	1	1	0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	0	1	1	0	1	0
Lens	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0

	Abrasive	Acoustic Insulator	Agitator	Airfoil	Ammeter	Analog Display	Armature	Battery	Bearing	Bell	Belt	Bladder	Blade	Brush	Burner	Buzzer	Cam	Cap	Capacitor	Carburetor	Carousel	Catalytic Converter	Centrifuge	Check Valve	Choke	Circuit Board	Clamp	Clutch	Condenser	Container	Conveyor	Cover	Crank	Cushion	Digital Display	Diode	Displacement Gauge	Diverter	Divider	Door	Electric Cord	Electric Distributor	Electric Insulator	Electric Motor	Electric Plate	Electric Plug	Electric Socket	Electric Switch	Electric Wire	Electrical Transformer	Electromagnet	EM Sensor	EM Transmitter	Evaporator	Fan	Flag	Flywheel	Friction Enhancer	Fuse	Gear	Generator	Glue	Handle	Heat Exchanger	Heating Element	Hinge	Housing	Hydraulic Piston	Hydraulic Pump	IC Motor	Inclined Plane	Indicator Light	Inductor	Insert	Key	Knob	Latch Release	Lens	Level Gauge	Lever	Light Source	Link	Mold
Level Gauge	0	0	1	0	0	1	0	0	1	1	1	0	0	0	0	0	1	0	0	0	1	0	1	0	0	0	0	1	0	1	1	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	1	1	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	1	0	1	1	0	1	0
Lever	1	0	0	1	0	1	1	0	1	1	1	0	0	1	0	0	1	0	0	0	1	0	1	0	0	0	1	1	0	0	1	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	1	1	0	1	0	0	0	0	1	1	1	1	0	0	0	0	0	1	0	1	1	0	1	0
Light Source	0	0	0	0	1	1	0	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	1	0	1	0	0	0	1	0	0	0	0	1	0	0	1	0	0	0	0	1	0	0	1	0	0
Link	1	0	0	1	0	1	1	0	1	1	1	0	0	1	0	0	1	0	0	0	1	0	1	0	0	0	1	1	0	0	1	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	1	1	0	1	0	0	0	0	1	1	1	1	0	0	0	0	0	1	0	1	1	0	1	0
Mold	0	0	1	0	0	1	0	0	0	1	0	1	1	0	0	0	0	1	0	1	1	0	1	1	1	0	0	0	1	1	1	1	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Needle	1	0	0	1	0	1	1	0	1	1	1	0	0	1	0	0	1	0	0	0	1	0	1	0	0	0	0	1	0	0	1	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	1	1	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	1	0	1	1	0	1	0
Nozzle	1	0	1	0	0	1	0	0	0	1	0	1	1	1	1	0	0	1	0	1	0	1	1	1	1	0	0	0	1	1	0	1	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1
Nut-Bolt	0	0	1	0	0	1	0	0	0	1	0	1	1	1	0	0	0	0	0	0	1	0	1	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Permeable Membrane	1	0	1	0	0	1	0	0	0	1	0	1	1	1	0	0	0	1	0	1	1	0	1	1	1	0	0	0	0	1	1	1	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Pipe	0	0	1	1	0	1	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	1	1	1	1	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Pneumatic Piston	1	0	0	0	0	1	0	0	1	1	1	0	0	1	0	0	1	0	0	1	1	1	0	1	1	0	0	1	1	1	1	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	1	0	1	0
Pneumatic Pump	0	0	0	1	0	1	1	0	1	1	1	0	0	0	1	0	1	0	0	1	0	1	0	1	1	0	0	1	1	1	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	0	1	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	1	0	1	0
Potentiometer	0	0	0	0	1	1	0	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	1	0	1	0	0	0	1	0	0	0	0	1	0	1	1	0	0	1	0	0	0	0	1	0	0
Pressure Gauge	0	0	0	1	0	1	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	1	1	1	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0
Pressure Vessel	0	0	1	0	0	1	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	1	1	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Pulley	1	0	0	1	0	1	1	0	1	1	1	0	0	1	0	0	1	0	0	0	1	0	1	0	0	0	0	1	0	0	1	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	1	1	0	1	0	0	0	0	1	1	1	1	0	0	0	0	0	1	0	1	1	0	1	0
Punch	0	0	1	0	0	1	0	0	0	1	0	1	1	1	0	0	0	0	0	0	1	0	1	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Rake	1	0	1	0	0	1	0	0	0	1	0	1	1	1	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Recording	0	1	0	0	1	1	0	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	1	0	1	0	0	0	1	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	1	0	0
Reservoir	1	0	1	0	0	1	0	0	0	1	0	1	1	1	1	0	0	1	0	1	1	0	1	1	1	0	0	0	1	1	1	1	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Resistor	0	0	0	0	1	1	0	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	1	0	1	0	0	0	1	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	1	0	0
Retaining Clip	0	0	1	0	0	1	0	0	0	1	0	1	1	1	0	0	0	0	0	0	1	0	1	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
Rivet	0	0	1	0	0	1	0	0	0	1	0	1	1	1	0	0	0	0	0	0	1	0	1	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
Rotational Coupler	1	0	0	1	0	1	1	0	1	1	1	0	0	1	0	0	1	0	0	0	1	0	1	0	0	0	1	1	0	0	1	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	1	1	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	1	0	1	1	0	1	0
Screen	1	0	1	0	0	1	0	0	0	1	0	1	1	1	0	0	0	1	0	1	1	0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Screw	0	0	1	0	0	1	0	0	0	1	0	1	1	1	0	0	0	0	0	0	1	0	1	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
Screw Propeller	0	0	0	1	0	1	1	0	1	1	1	0	0	0	0	0	1	0	0	1	0	0	1	0	0	0	0	1	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	0	0	1	0	0	0	1	1	1	1	0	0	0	0	0	1	0	1	1	0	1	1
Seal	1	0	1	0	0	1	0	0	0	1	0	1	1	1	0	0	0	1	0	1	1	1	1	1	1	0	0	0	1	1	1	1	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Shaft	1	0	0	1	0	1	1	0	1	1	1	0	0	1	0	0	1	0	0	0	1	0	1	0	0	0	0	1	0	0	1	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	1	1	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	1	0	1	1	0	1	0
Signal Filter	0	0	0	0	1	1	0	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	1	0	1	0	0	0	1	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	1	0	0
Sled	1	0	0	1	0	1	1	0	1	1	1	0	0	1	0	0	1	0	0	0	1	0	1	0	0	0	1	1	0	0	1	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	1	1	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	1	0	1	1	0	1	0
Solder	0	0	0	0	1	1	0	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	1	0	1	0	0	0	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	1	0	0
Speaker	0	1	0	0	1	1	0	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	1	0	1	0	0	0	1	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	1	0	0
Speed Gauge	0	0	0	1	0	1	1	0	1	1	1	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	0	0	1	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	1	1	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	1	1	1	1	0	1	0
Spring	1	0	0	1	0	1	1	0	1	1	1	0	0	1	0	0	1	0	0	0	1	0	1	0	0	0	1	1	0	0	1	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	1	1	0	1	0	0	0	0	1	1	1	1	0	0	0	0	0	1	0	1	1	0	1	0
Sprocket	1	0	0	1	0	1	1	0	1	1	1	0	0	1	0	0	1	0	0	0	1	0	1	0	0	0	0	1	0	0	1	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	1	1	0	1	0	0	0	0	1	1	1	1	0	0	0	0	0	1	0	1	1	0	1	0
Stop	1	0	0	1	0	1	1	0	1	1	1	0	0	1	0	0	1	0	0	0	1	0	1	0	0	0	1	0	0	0	1	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	1	1	0	1	0	0	0	0	1	1	1	1	0	0	0	0	0	1	0	1	1	0	1	0
Stuffing	0	0	1	0	0	1	0	0	0	1	0	1	1	1	0	0	0	0	0	0	1	0	1	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Support	0	0	1	0	0	1	0	0	0	1	0	1	1	1	0	0	0	0	0	0	1	0	1	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
Thermal Insulator	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Thermal Plate	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Thermal Wire	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Thermostat	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Transistor	0	0	0	0	1	1	0	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	0	0	1	0	1	1	0	0	1	0	0	0	0	1	0	1	1	0	0	1	0	0	0	0	1	0	0
Tube	1	0	1	0	0	1	0	0	0	1	0	1	1	1	0	0	0	1	0	1	1	1	1	1	1	0	0	0	1	1	1	1	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Valve	1	0	1	0	0	1	0	0	0	1	0	0	0	1	1	0	0	1	0	1	0	1	1	1	1	0	0	0	1	1	0	1	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1	1	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	1
Varistor	0	0	0	0	1	1	0	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	1	0	1	1	0	0	1	0	0	0	0	1	0	1	1	0	0	1	0	0	0	0	1	0	0
Vibrator	0	1	1	0	0	1	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0	1	0	1	1	0	1	0
Voltmeter	0	0	0	0	1	1	0	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	1	0	1	1	0	0	1	0	0	0	0	1	0	1	1	0	0	0	0	1	0	0	1	0	0
Washer	0	0	1	0	0	1	0	0	0	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
Wheel	1	0	0	1	0	1	1	0	1	1	1	0	0	1	0	0	1	0	0	0	1	0	1	0	0	0	1	1	0	0	1	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0	1	0	1	1	0	1	0

	Needle	Nozzle	Nut-Bolt	Permeable Membrane	Pipe	Pneumatic Piston	Pneumatic Pump	Potentiometer	Pressure Gauge	Pressure Vessel	Pulley	Punch	Rake	Recording	Reservoir	Resistor	Retaining Clip	Rivet	Rotational Coupler	Screen	Screw	Screw Propeller	Seal	Shaft	Signal Filter	Sled	Solder	Speaker	Speed Gauge	Spring	Sprocket	Stop	Stuffing	Support	Thermal Insulator	Thermal Plate	Thermal Wire	Thermostat	Transistor	Tube	Valve	Varistor	Vibrator	Voltmeter	Washer	Wheel
Abrasive	1	1	0	1	0	1	0	0	0	0	1	0	1	0	1	0	0	0	1	1	0	0	1	1	0	1	0	0	0	1	1	1	0	0	0	0	0	0	0	1	1	0	0	0	0	1
Acoustic Insulator	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
Agitator	0	1	1	1	1	0	0	0	0	1	0	1	1	0	1	0	1	1	0	1	1	0	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1	1	0	1	0	1	0
Airfoil	1	0	0	0	1	0	1	0	1	0	1	0	0	0	0	0	0	0	1	0	0	1	0	1	0	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Ammeter	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0
Analog Display	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	1	1	1	1	1	1	1	1
Armature	1	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	1	0	1	0	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Battery	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0
Bearing	1	0	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	1	0	1	0	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Bell	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	1	1	1	1	1	1	1	1
Belt	1	0	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	1	0	1	0	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Bladder	0	1	1	1	0	0	0	0	0	0	0	1	1	0	1	0	1	1	0	1	1	0	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1	0	0	1	0	1	0
Blade	0	1	1	1	0	0	0	0	0	0	0	1	1	0	1	0	1	1	0	1	1	0	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1	0	0	1	0	1	0
Brush	1	1	1	1	0	1	0	0	0	0	1	1	1	0	1	0	1	1	1	1	1	0	1	1	0	1	0	0	0	1	1	1	1	1	0	0	0	0	0	1	1	0	0	0	1	1
Burner	0	1	0	0	1	0	1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	1	0	0	0	0	0
Buzzer	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0
Cam	1	0	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	1	0	1	0	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Cap	0	1	0	1	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
Capacitor	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0
Carburetor	0	1	0	1	1	1	1	0	1	1	0	0	1	0	1	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
Carousel	1	0	1	1	0	1	0	0	0	0	1	1	1	0	1	0	1	1	1	1	1	0	1	1	0	1	0	0	1	1	1	1	1	1	0	0	0	0	0	1	0	0	0	0	0	1
Catalytic Converter	0	1	0	0	1	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
Centrifuge	1	1	1	1	1	0	0	0	1	1	1	1	0	0	1	0	1	1	1	1	1	1	1	1	0	1	0	0	0	1	1	1	1	1	0	0	0	0	0	1	1	0	1	0	0	1
Check Valve	0	1	0	1	1	1	1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
Choke	0	1	0	1	1	1	1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
Circuit Board	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0
Clamp	0	0	1	0	1	0	0	0	0	0	0	1	0	0	0	0	1	1	1	0	1	0	0	0	0	1	0	0	0	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	1
Clutch	1	0	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	1	0	1	0	1	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Condenser	0	1	0	0	1	1	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
Container	0	1	1	1	1	1	1	0	1	1	0	1	0	0	1	0	1	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1	1	0	1	0	0	0
Conveyor	1	0	1	1	0	1	0	0	0	0	1	1	1	0	1	0	1	1	1	1	1	0	1	1	0	1	0	0	1	1	1	1	1	1	0	0	0	0	0	1	0	0	0	0	0	1
Cover	0	1	0	1	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
Crank	1	0	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	1	0	1	0	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Cushion	1	0	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	1	0	1	0	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Digital Display	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0
Diode	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0
Displacement Gauge	1	0	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	1	0	1	0	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Diverter	0	1	0	1	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
Divider	0	1	0	1	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
Door	0	1	0	1	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
Electric Cord	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0
Electric Distributor	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0
Electric Insulator	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0
Electric Motor	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0
Electric Plate	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0
Electric Plug	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0
Electric Socket	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0
Electric Switch	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0
Electric Wire	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0
Electrical Transformer	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0
Electromagnet	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0
EM Sensor	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0
EM Transmitter	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0
Evaporator	0	1	0	0	1	1	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
Fan	1	0	0	0	1	1	0	0	1	1	1	0	0	0	1	0	0	0	1	0	0	0	1	1	0	1	0	0	1	1	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	1
Flag	1	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Flywheel	1	0	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	1	0	1	0	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1
Friction Enhancer	1	0	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	1	0	1	0	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1
Fuse	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0
Gear	1	0	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	1	0	1	0	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1
Generator	1	0	0	0	0	1	0	1	0	0	1	0	0	1	0	1	0	0	1	0	0	0	0	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	1	0	0	1	0	1	0	1
Glue	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	1	1	1	1	1	1	1
Handle	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	1	1	0	1	0	0	1
Heat Exchanger	0	1	0	0	0	0	1	0	1	1	0	0	0	0	1	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	1	1	0	0	0	0	0
Heating Element	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	1	1	1	1	1	0	0	1	0	1	0	0
Hinge	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0
Housing	0	0	1	0	0	0	0	0	0	0	0	1	0	0	1	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0
Hydraulic Piston	1	0	0	0	1	0	0	0	1	1	1	0	0	0	0	0	0	0	1	0	0	1	0	1	0	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	1	0	1	0	0	1
Hydraulic Pump	1	0	0	0	1	0	0	0	1	1	1	0	0	0	0	0	0	0	1	0	0	1	0	1	0	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	1	0	1	0	0	1
IC Motor	1	0	0	0	0	0	0	1	0	0	1	0	0	1	0	1	0	0	1	0	0	1	0	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	1	0	0	1	1	1	0	1
Inclined Plane	1	0	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	1	0	1	0	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1
Indicator Light	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0
Inductor	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0
Insert	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0
Key	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0
Knob	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0
Latch Release	1	0	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	1	0	1	0	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1
Lens	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0

	Needle	Nozzle	Nut-Bolt	Permeable Membrane	Pipe	Pneumatic Piston	Pneumatic Pump	Potentiometer	Pressure Gauge	Pressure Vessel	Pulley	Punch	Rake	Recording	Reservoir	Resistor	Retaining Clip	Rivet	Rotational Coupler	Screen	Screw	Screw Propeller	Seal	Shaft	Signal Filter	Sled	Solder	Speaker	Speed Gauge	Spring	Sprocket	Stop	Stuffing	Support	Thermal Insulator	Thermal Plate	Thermal Wire	Thermostat	Transistor	Tube	Valve	Varistor	Vibrator	Voltmeter	Washer	Wheel
Level Gauge	1	1	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	1	0	1	0	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1
Lever	1	0	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	1	0	1	0	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1
Light Source	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0
Link	1	0	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	1	0	1	0	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1
Mold	0	1	0	1	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
Needle	1	0	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	1	0	1	0	1	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1
Nozzle	0	1	0	1	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	0	0
Nut-Bolt	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
Permeable Membrane	0	1	0	1	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	0	0
Pipe	0	0	0	0	1	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Pneumatic Piston	1	0	0	0	1	1	1	0	1	1	1	0	0	0	0	0	0	0	1	0	0	1	0	1	0	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1
Pneumatic Pump	1	0	0	0	1	1	1	0	1	1	1	0	0	0	0	0	0	0	1	0	0	1	0	1	0	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1
Potentiometer	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0
Pressure Gauge	0	0	0	0	1	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Pressure Vessel	0	0	0	0	1	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Pulley	1	0	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	1	0	1	0	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1
Punch	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
Rake	0	1	0	1	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	0	0
Recording	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0
Reservoir	0	1	0	1	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	0	0
Resistor	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0
Retaining Clip	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0
Rivet	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0
Rotational Coupler	1	0	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	1	0	1	0	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1
Screen	0	1	0	1	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	0	0
Screw	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0
Screw Propeller	1	0	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	1	0	1	0	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Seal	0	1	0	1	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	0	0
Shaft	1	0	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	1	0	1	0	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1
Signal Filter	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0
Sled	1	0	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	1	0	1	0	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1
Solder	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0
Speaker	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0
Speed Gauge	1	0	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	1	0	1	0	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1
Spring	1	0	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	1	0	1	0	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1
Sprocket	1	0	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	1	0	1	0	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1
Stop	0	0	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	1	0	1	0	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1
Stuffing	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	1	0	1	0	0	0
Support	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	1
Thermal Insulator	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
Thermal Plate	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
Thermal Wire	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
Thermostat	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
Transistor	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0
Tube	0	1	0	1	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	1	0	1	0	0	0
Valve	0	1	0	1	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	1	0	1	0	0	0
Varistor	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0
Vibrator	1	1	0	1	0	1	1	0	0	0	1	0	1	0	1	0	0	0	1	1	0	0	1	1	0	1	0	0	1	1	1	1	1	0	0	0	0	0	0	1	1	0	1	0	0	1
Voltmeter	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0
Washer	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0
Wheel	1	0	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	1	0	1	0	1	0	0	1	1	1	1	0	1	0	0	0	0	0	0	0	0	1	0	0	1

REFERENCES

Al-Hakim, L., Kusiak, A. and Mathew, J., 2000. “A Graph-Theoretic Approach to Conceptual Design with Functional Perspectives,” *Computer-Aided Design*, 32 (14): 867–875.

Altschuller G., 1984. *Creativity as an Exact Science*, New York, New York, Gordon and Breach.

Akman, V., ten Haagen, P.J.W., and Tomiyama, T., 1990. “A Fundamental and Theoretical Framework for an Intelligent CAD System,” *Computer-Aided Design*, 22(6): 352–368.

Anderson, J.R., 1981. *Cognitive Skills and Their Acquisition*, Hillsdale, NJ, Lawrence Erlbaum Association.

Andrews P. and Snowden D., 2002. “Next Generation Knowledge Management: The Complexity of Humans,” Executive Tek Report, IBM Global Services.

Antonsson, E. K., Cagan, J., 2001. *Formal Engineering Design Synthesis*, Cambridge University Press.

Bhadra, A. and Fischer, G.W., 1988. “A New GT Classification Approach: A Database with Graphical Dimensions,” *Manufacturing Review*, 1(1): 44–49.

Bohm, M., and Stone, R., 2004. “Representing Functionality to Support Reuse: Conceptual and Supporting Functions,” Proceedings of DETC’04, Salt Lake City, UT, DETC2004-57693.

Bohm M.R., Stone R.B., Szykman S., 2004. "Representing Functionality to Support Reuse: Conceptual and Supporting Functions," Proceedings of ASME DETC and CIE Conferences, September 28-October 2, Salt Lake City, Utah, DETC2004-57693.

Bohm, M., Stone, R. and Szykman, S., 2005. “Enhancing Virtual Product Representations for Advanced Design Repository Systems,” *Journal of Computer Information Science in Engineering*, 5(4): 360–372.

Black, I., 1990. “Embodiment Design: Facilitating a Simultaneous Approach to Mechanical CAD,” *Computer-Aided Engineering Journal*, 7: 49–53.

Bracewell, R.H. and Sharpe, J.E.E., 1996. “Functional Descriptions Used in Computer Support for Qualitative Scheme Generation–’Schemebuilder’,” *Artificial Intelligence for Engineering Design, Analysis and Manufacture (AIEDAM)*, 10(4): 333–345.

Bradley, D.A., Bracewell, R.H., and Chaplin, R.V., 1993. “Engineering Design and Mechatronics: The Schemebuilder Project,” *Research in Engineering Design*, 4: 241–248.

Campbell, M.I., Cagan, J., Kotovsky, K., 1999. "A-Design: An Agent-Based Approach to Conceptual Design in a Dynamic Environment," *Research in Engineering Design*, 2: 172–192.

Campbell, M.I., Cagan, J., Kotovsky, K., 2000. "Agent-Based Synthesis of Electro-mechanical Design Configurations," *Journal of Mechanical Design*, 122: 61–69.

Campbell, M.I., Cagan, J, and Kotovsky, K., 2001. “Learning From Design Experience: Todo/Taboo Guidance,” Proceedings of the 2001 ASME Design Engineering Technical Conferences and Computers in Engineering Conference, DETC01/ DTM-21687, Pittsburgh, PA.

Campbell M.I., Cagan J., Kotovsky K., 2003. "The A-Design Approach to Managing Automated Design Synthesis," *Research in Engineering Design*, 14: 12–24.

Cera, C.D., Regli, W.C., Braude, I., Shapirstein, Y. and Foster, C.V., 2002. “A Collaborative 3D Environment for Authoring Design Semantics,” *IEEE Computer Graphics and Applications: Graphics in Advanced Computer-Aided Design*, May/June: 43–55.

Chakrabarti, A. and Bligh, T.P., 2001. “A Scheme for Functional Reasoning in Conceptual Design,” *Design Studies*, 22(6): 493–516.

Chenhall, R. G., 1978. *Nomenclature for Museum Cataloging: A System for Classifying Man-Made Objects*, American Association for State and Local History, Nashville, TN.

Cutherell, D., 1996. “Chapter 16: Product Architecture,” *The PDMA Handbook of New Product Development*, M. Rosenau Jr., *et al.*, ed., John Wiley and Sons.

Cross, N., 1994. *Engineering Design Methods: Strategies for Product Design*, 2nd Ed., John Wiley and Sons, Chichester, UK.

Culley, S. J., and Webber, S. J., 1992. “Implementation Requirements for Electronic Standard Component Catalogues,” *Proceedings Institution of Mechanical Engineers, Journal of Engineering Manufacture: Part B*, 206: 253–260.

de Bono, E., 1970. *Lateral Thinking: Creativity Step by Step*, Harper and Row, Publishers, Inc. New York, NY.

Deng, Y.M., 2002. “Function and Behavior Representation in Conceptual Mechanical Design,” *Artificial Intelligence for Engineering Design, Analysis and Manufacture (AIEDAM)*, 16: 343–362.

Du, X. and Chen, W., 2004. "Sequential Optimization and Reliability Assessment for Probabilistic Design," *Journal of Mechanical Design*, 126(2): 225–233.

Eggli, L., Ching-yao, H., Bruderlin, B.D. and Elber, G., 1997. “Inferring 3D Models from Freehand Sketches and Constraints,” *Computer-Aided Design*, 29(2): 101–112.

Feng, C.X., Li, P.G., and Liang, M., 2001. “Fuzzy Mapping of Requirements onto Functions in Detail Design,” *Computer-Aided Design*, 33: 425–437.

Finkelstein, L., El-hami, M., and Ginger, R., 1998. “Conceptual Design of Instrument Systems Utilising Physical Laws,” *Measurement*, 23: 9–13.

Fox, E.P., 1994. “The Pratt and Whitney Probabilistic Design System,” 35th AIAA/ ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, AIAA-94-1442-CP, April.

Girdhar, A. and Mital, A., 2001a. “Expanding Group Technology Part Coding for Functionality: Part I–Developing a Functional Basis for Classification,” *International Journal of Industrial Engineering–Applications and Practice*, 8(3): 186–197.

Girdhar, A. and Mital, A., 2001b. “Expanding Group Technology Part Coding for Functionality: Part II–Functional Classification of Workparts and Application to Design,” *International Journal of Industrial Engineering–Applications and Practice*, 8 (3): 198–209.

Glover J., Ronning R. and Reynolds C., (eds.), 1989. *Handbook of Creativity*, Plenum, London, UK.

Gorti, S.R. and Sriram, R.D., 1996. “ From Symbol to Form: A Framework for Conceptual Design,” *Computer-Aided Design*, 28(11): 853–870.

Greer J.L., Stock M.E., Stone R.B., Wood K.L., 2003. "Enumerating the Component Space: First Steps toward a Design Naming Convention for Mechanical Parts," Proceedings of ASME DETC and CIE Conferences, September 2-6, 2003, Chicago, Illinois, DETC03/DTM-48666.

Gruber, T. R., 1994. “Toward Principles for the Design of Ontologies Used for Knowledge Sharing,” *International Journal of Human Computer Studies*, 43(5/6): 907–928.

Gui, J.K. and Mäntylä, M., 1994. “Functional Understanding of Assembly Modelling,” *Computer-Aided Design*, 26(6): 435-451.

Guilford, J.P., 1959. *‘Traits of Creativity’*, in P.E. Vernon (ed.), Creativity, Penguin, Harmondsworth.

Hayes, C.C., 1995. “Use of Function Information to Create Redesign Suggestions,” Proceedings of the 8th Florida Artificial Intelligence Research Symposium (FLAIRS), 309–313.

Hearst, M.A., Gross, M.D., Landay, J.A. and Stahovich, T.F., 1998. “Sketching Intelligent Systems,” *IEEE Intelligent Systems*, 13(3): 10–19.

Henderson, M. and Musti, S., 1988. “Automated Group Technology Part Coding from a Three-Dimensional CAD Database,” *Journal of Engineering in Industry*, 110(3): 278–287.

Hennig, Willi, 1979. *Phylogenetic systematics* (tr. D. Dwight Davis and Rainer Zangerl). Urbana, IL: Univ. of Illinois Press (reprinted 1999).

Hicks, B. J., Culley, S. J., and Mullineux, G., 2005. “The Modeling of Engineering Systmes for their Computer Based Embodiment with Standard Components,” *Journal of Mechanical Design*, 127, pp. 414–432.

Hirtz, J.M., Stone, R.B., McAdams, D.A., Szykman, S., Wood, K.L., 2002. "A Functional Basis for Engineeering Design: Reconciling and Evolving Previous Efforts," *Research in Engineering Design*, 13(2), pp. 65–82.

Homem de Mello, L.S. and Sanderson, A.C., 1991. “A Correct and Complete Algorithm for the Generation of Mechanical Assembly Sequences,” *IEEE Transactions on Robotics and Automation*, 7(2): 228–240.

Hubka, V. and Eder, W., 1984. *Theory of Technical Systems*, Springer-Verlag, Berlin.

Hwang, T. and Ullman, D., 1990. “The Design Capture System: Capturing Back-of-the-Envelope Sketches,” *Journal of Engineering Design*, 1(4): 339–353.

Ishii, K., Adler, R., and Barkan, P., 1988. “Application of Design Compatibility Analysis to Simultaneous Engineering,” *Artificial Intelligence for Engineering Design and Manufacture (AIEDAM)*, 2(1): 53–65.

Ivashkok, M., 2004. *ACCEL: a Tool Supporting Concept Generation in the Early Design Phase*. PhD thesis, The Eindhoven University of Technology, Eindhoven, The Netherlands.

Johnson, A., 1998. “An Open Architecture Approach to Kinematic Analysis for Computer-Aided Embodiment Design,” *Computer-Aided Design*, 30(3): 199–204.

Jordan, R., Van Wie, M., Stone, R.B., Wang, J., and Terpenny, J., 2005. “A Group Technology Based Representation for Product Portfolios,” Proceedings of IDETC/CIE 2005, DETC2005-85313, Long Beach, CA.

Kitamura, Y. and Mizoguchi, R., 2003. “Ontology-based Description of Functional Design Knowledge and its Use in a Functional Way Server,” *Expert Systems with Application*, 24(2): 153–166.

Kitamura, Y., Kashiwase, M., Fuse, M., and Mizoguchi, R., 2004. “Deployment of an Ontological Framework of Functional Design Knowledge,” *Advanced Engineering Informatics*, 18: 115–127.

Kurfman, M.A., Stone, R.B., Rajan, J.R., Wood, K.L., 2001. "Functional Modeling Experimental Studies," Proceedings of ASME DETC and CIE Conferences, September 9-12, Pittsburgh, Pennsylvania, DETC2001/DTM-21709.

Kurtoglu, T., Campbell, M.I., 2005. “Automated Synthesis of Elctromechanical Design Configurations from Empirical Analysis of Function to Form Mapping.” *Research in Engineering Design*, (In review).

Kurtoglu, T., Campbell, M.I., Bryant, C.R., Stone, R.B., McAdams, D.A., 2005. “Deriving a Component Basis for Computational Functional Synthesis. Proceedings of International Conference on Engineering Design,” ICED05, August 15-18, Melbourne, Australia.

Kurtoglu, T., Campbell, M.I., Bryant, C.R., Stone, R.B., 2007. “A Component Taxonomy as a Framework for Computational Design Synthesis,” *Journal of Computer and Information Science in Engineering,* (Accepted for publication).

Kusiak, A. and Szczerbicki, E., 1993. “Transformation from Conceptual to Embodiment Design,” *IIE Transactions*, 25(4): 6–11.

Kusiak, A., Szczerbicki, E., and Vujosevic, R., 1991. “Intelligent Design Synthesis: An Object-Oriented Approach,” *International Journal of Production Research*, 29(7): 1291–1308.

Liang V, Paredis, C.J.J., 2004. “A Port Ontology for Conceptual Design of Systems,” *Journal of Computing and Information Science in Engineering*, 4: 206–217.

Linnaei, Caroli, 1937. *Determinationes In Hortum Siccum Joachimi Burseri: the text of the manuscript in the Linnaean collections*, ed. Spencer Savage, Printed for the Linnaean Society by Taylor and Francis, London.

Linsey, J.S.., Green, M.G., Murphy, J.T., Wood, K.L., Markman, A.B., 2005. "Collabrating to Success: An Experimental Study of Group Idea Generation Techniques," Proceedings of DETC2005, Sept. 24-28, Long Beach, California.

Lipson, H. and Shpitalni, M., 2000. "Conceptual Design and Analysis by Sketching," *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing (AIEDAM)*, 14(5): 391–402.

Lu, C. and Russomanno, D.J., 1999. "KAT: A Knowledge Acquisition Tool for Acquiring Functional Knowledge Based Upon the No-Causality-in-Function Principle," Proceedings of the 1999 ACM Symposium on Applied Computing, 8–13.

Malmqvist, J. and Svensson, D., 1999. "A Design Theory Based Approach Towards Including QFD Data in Product Models," Proceedings of the 1999 ASME Design Engineering and Technical Conferences, September12-15, Las Vegas, Nevada.

Mann, D., 2000. "Towards a Generic Systematic Problem Solving and Innovation Design Methodology," Proceedings of the 2000 ASME Design Engineering and Technical Conferences, September 10-13, Baltimore, Maryland.

McAdams, D. and Wood, K., 2000. "Quantitative Measures for Design By Analogy," DETC2000/DTM-14562, Proceedings of DETC2000, Balitmore, MD.

Merriam-Webster Online, www.Merriam-Webster.com, copyright 2005 by Merriam-Webster, Incorporated.

Mittal, S., Dym, C., and Morjara, M., 1985. "PRIDE: An Expert system for the Design of Paper Handling Systems," *IEEE Computer*, 19(7): 102–114.

Moore, C.J., Miles, J.C., Rees, D.W.G., 1997. "Decision Support for Conceptual Bridge Design," *Artificial Intelligence in Engineering*, 11(3): 259–272.

Navinchandra D., Sycara K.P., Narasimhan S., 1991. “Behavioral Synthesis in CADET, a Case-Based Design Tool,” Proceedings Seventh IEEE Conference on Artificial Intelligence Applications, February,

Neches, R., Fikes, R., Finin, T., Gruber, T., Patil, R., Senator, T. and Swartout, W. R., 1991, “Enabling technology for Knowledge Sharing,” *AI Magazine*, 36–56.

Oh, V.K., Chaplin, R.V., Yan, X.T., Sharpe, J.E.E., 1996. “A Generic Framework for the Description of Components in the Design and Simulation of Mechatronic Products,” *Mechatronics–The Basis for New Industrial Development*, Southampton, Boston, Computational Mechanics Publications, 515–520.

Onyebueke, L.C., Onwubiko, C., and Chen, F.C., 1995. “Probabilistic Design Methodology and the Application of Probabilistic Fault Tree Analysis to Machine Design,” Proceedings of the 1995 ASME Design Engineering Technical Conferences, DE-Vol. 83.

Opitz, H., 1970. *A Classification System to Describe Workpiece,*. Pergamon Press, Ltd., Oxford, New York.

Opitz, H,, Eversheim, W,, and Wiendahl, H.P., 1969. “Workpiece Classification and its Industrial Application,” *International Journal of Machine Tool Design Research*, 9: 39–50.

Opitz, H. and Wiendahl, H.P., 1971. “Group Technology and Manufacturing Systems for Small and Medium Quantity Production,” *International Journal of Production Research*, 9 (1): 181–203.

Osborn, A., 1957. *Applied Imagination*, Scribner, New York, NY.

Otto, K. and Wood, K., 1996. “A Reverse Engineering and Redesign Methodology for Product Evolution,” Proceedings of the 1996 ASME Design Theory and Methodology Conference, Irvine, CA, DETC96/DTM-1523.

Otto, K. and Wood, K., 1997, “Conceptual and Configuration Design of Products and Assemblies,” *ASM Handbook, Materials Selection and Design*, 20, ASM International.

Otto, K. and Wood, K., 2001. *Product Design*. Prentice Hall, Upper Saddle River, NJ.

Pahl, G., and Beitz, W., 1996. *Engineering Design—A Systematic Approach*, 2nd edition, Springer, London.

Palmer, R.S. and Shapiro, V., 1993. “Chain Models of Physical Behavior for Engineering Analysis and Design,” *Research in Engineering Design*, 5: 161–184.

Paredis, C. J. J., Diaz-Calderon, A., Sinha, R., Khosla, P. K., 2001. “Composable Models for Simulation-Based Design,” *Engineering with Computers*, 17: 112–128, Springer-Verlag, London.

Pimmler, T. and Eppinger, S., 1994, “Integration Analysis of Product Decompositions,” Proceedings of the ASME Design Theory and Methodology Conference, 68.

Prasad B., 1998. “Review of QFD and Related Deployment Techniques,” *Journal of Manufacturing Systems*, 17(3): 221–235.

Pugh, S., 1991. *Total Design*, Addison Wesley, Wokingham, UK.

Qin, S.F., Wright, D.K. and Jordano, I.N., 2000. “From On-line Sketching to 2D and 3D Geometry: A System Based on Fuzzy Knowledge,” *Computer-Aided Design*, 32 (14): 851–866.

Radcliffe, D. and Lee, T.Y., 1989. “Design Methods Used by Undergraduate Engineering Students,” *Design Studies*, 10(4): 199–207.

Rohrbach, B., 1969. “Kreativ nach Regeln – Methode 635, eine Neue Technik zum Lösen von Problemen,” *Absatzwirtschaft*, 12: 73–75.

Roozenburg, N.F.M. and Eekels, J., 1995. *Product Design: Fundamentals and Methods*, Wiley, Chichester, UK.

Schmidt, L. and Cagan, J., 1995. “Recursive Annealing: A Computational Model for Machine Design,” *Research in Engineering Design*, 7(2): 102–125.

Schmidt, L. and Cagan, J., 1997. “GGREADA: A Graph Grammar-Based Machine Design Algorithm,” *Research in Engineering Design*, 9(4): 195–213.

Serran, D. and Gossard, D., 1992. “Tools and Techniques for Conceptual Design,” *Artificial Intelligence in Engineering Design: Design Representation and Models of Routine Design*, Academic Press, San Diego, 71–116.

Shah, J. J., 1998. “Experimental Investigation of Progressive Idea Generation Techniques in Engineering Design,” Proceedings of the DETC’98, 1998 ASME Design Engineering Technical Conferences, Atlanta, GA.

Shah, J. and Bhatnagar, A., 1989. “Group Technology Classification from Feature Based Geometric Models,” *Manufacturing Review*, 2(3): 204–213.

Shimomura, Y., Tanigawa, S., Takeda, H., Umeda, Y. and Tomiyama, T., 1996. “Functional Evaluation Based on Function Content,” Proceedings of the 1996 ASME Design Theory and Methodology Conference, Irvine, CA, DETC96/ DTM-1532.

Sieger, D.B. and Salmi, R.E., 1997. “Knowledge Representation Tool for Conceptual Development of Product Designs,” IEEE International Conference on Systems, Man, and Cybernetics ‘Computational Cybernetics and Simulation,’ 2: 1936–1941.

Simpson, T.W., Bauer, M.D., Allen, J.K., and Mistree, F., 1995. “Implementation of DFA in Conceptual and Embodiment Design Using Decision Support Problems,” Proceedings of the 1995 ASME Design Engineering Technical Conferences, DE-Vol. 82.

Sridharan P., Campbell M.I., 2005. "A Study on the Grammatical Construction of Function Structures," *Artificial Intelligence in Engineering Design, Analysis, and Manufacture,* 19: 139–160.

Stahovich, T. F., Davis, R., Shrobe, H., 1993. “An Ontology of Mechanical Devices,” AAAI-93, Working Notes, Reasoning About Function, 137–140.

Stone, R. and Wood, K., 2000. “Development of a Functional Basis for Design,” *Journal of Mechanical Design*, 122(4): 359–370.

Strawbridge, Z., McAdams, D. A. and Stone, R. B., 2002, “A Computational Approach to Conceptual Design,” DETC02/DTM-34001, Proceedings of DETC2002, ASME, Montreal, Canada.

Sturgill, M., Cohen, E. and Riesenfel, R.F., 1995. “Feature-Based 3-D Sketching for Early Stage Design,” Proceedings of the ASME Computers in Engineering Conference, ASME Press, New York, 545–552.

Subramanian, D. and Wang, C.S., 1995. “Kinematic Synthesis with Configuration Spaces,” *Research in Engineering Design*, 7(3): 193–213.

Suh N.P., 1995. "Axiomatic Design of Mechanical Systems," *Journal of Mechanical Design*, 117: 2–10.

Thornton, A.C. and Johnson, A.L., 1996. “CADET: A Software Support Tool for Constraint Processes in Embodiment Design,” *Research in Engineering in Design*, 8: 1–13.

Ullman, D., 1997. *The Mechanical Design Process,* 2nd Ed., McGraw-Hill.

Ulrich, K. and Eppinger, S., 1995. *Product Design and Development*, McGraw-Hill.

Ulrich, K. and Seering, W., 1988. “Computation and Conceptual Design,” *Robotics and Computer-Integrated Manufacturing,* 4(3/4): 309–315.

Umeda, Y., Ishii, M., Yoshioka, M., Shimomura, Y., and Tomiyama, T., 1996. “Supporting Conceptual Design Based on the Function-Behavior-State Modeler,” *Artificial Intelligence for Engineering Design, Analysis and Manufacture (AIEDAM)*, 10(4): 275–288.

Uschold, M., 1998. “Knowledge level modeling: Concepts and terminology.,” *Knowledge Engineering Review*, 13(1). Also available as AIAI-TR-196 from AIAI, The University of Edinburgh.

van den Berg, N., Dutilh, C., Huppes, G., 1995. “Beginning LCA: A Guide into Environmental Life Cycle Assessment,” *Center of Environmental Science,* Leiden University, Unilever and CML.

Wallace, A. P., 1995. *The Modelling of Engineering Assemblies Based on Standard Catalogue Components,* University of Bath, UK.

Ward, A., 1989. *A Theory of Quantitative Inference Applied to a Mechanical Design Compiler*, Dissertation, Massachusetts Institute of Technology.

Ward, A. and Seering, W., 1993. “Quantitative Inference in a Mechanical Design 'Compiler',” *Journal of Mechanical Design*, 115: 29-35.

Welch, R.V. and Dixon, J.R., 1991. “Conceptual Design of Mechanical Systems,” Proceedings of the Design Theory and Methodology Conference, DE-Vol. 31.

Williams, B.C., 1990. “Interaction-based invention: designing novel devices from first principles,” AAAI-90 Proceedings. Eighth National Conference on Artificial Intelligence, Boston, MA, 1: 349–356.

Wodehouse, A., Grierson, H., Ion, W.J., Juster, N., Lynn, A. and Stone, A.L., 2004. “Tikiwiki: A Tool to Support Engineering Design Students in Concept Generation,” International Engineering and Product Design Education Conference, Delft, the Netherlands, September 2-3 2004.

Yang, M.C., 2003. “Concept Generation and Sketching: Correlations with Design Outcome.” Proceedings of ASME Design Engineering Technical Conferences, Chicago, IL, September 2-6, DETC2003/DTM-48677.

Yates, III, W.D. and Beaman, D.M., 1995. “Design Simulation to Improve Product Reliability,” Proceedings of the Annual Reliability and Maintainability Symposium, 193–199.

Zha, X.F., Du, H.J., and Qiu, J.H., 2001. “Knowledge-Based Approach and System for Assembly Oriented Design, Part I: The Approach,” *Engineering Applications of Artificial Intelligence*, 14(1): 61–75.

Zwicky, F., 1969. *Discovery, Invention, Research-Through the Morphological Approach*, The Macmillian Company, Toronto.

VITA

Cari Rihan Bryant was born on March 3, 1977 to F. Allen and G. Deon Bryant in Wichita, Kansas and was raised in Kansas City, Missouri. After graduating from Raytown High School in 1995, she attended the University of Missouri–Rolla. In 2000, she received her Bachelor of Science degree in mechanical engineering from UMR and entered the Master's Degree program in mechanical engineering at the University of Michigan in Ann Arbor, Michigan. In 2003, after earning a Master's Degree in mechanical engineering and a Master's Degree in biomedical engineering from UMich, she returned to the University of Missouri–Rolla to work on a doctoral degree in mechanical engineering. She completed her Ph.D. in July of 2007 and joined the faculty at the Pennsylvania State University in August 2007 as an Assistant Professor of Engineering Design and Mechanical Engineering, where she was awarded the James F. Will Career Development Professorship.