2006-05-04

Analogical Matching Using Device-Centric and Environment-Centric Representations of Function

Greg P. Milette
Worcester Polytechnic Institute

Analogical Matching Using Device-Centric and Environment-Centric Representations of Function

by

Greg Milette

A Thesis

Submitted to the Faculty

of

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfilment of the requirements for the

Degree of Master of Science

in

Computer Science

May 2006

Approved:

_____

Dr. David C. Brown, Advisor

_____

Dr. George Heineman, Reader

_____

Dr. Michael A. Gennert, Department Head

Abstract

Design is hard and needs to be supported by software. One of the ways software can support designers is by providing analogical reasoning. To make analogical reasoning work well, the software makers need to know how to create a knowledge representation that will facilitate the kind of analogies that the designers want. This thesis will inform software makers by experimenting with two kinds of knowledge representations, called device-centric (DC) and environment-centric (EC), and to try to determine the relative benefits of using either one of them for analogical matching. We performed computational experiments, using Structure Mapping Engine for matching, to determine the quantity and quality of analogical matches that are produced when the representation is varied. We conducted a limited human experiment, using questionnaires and repertory grids, to determine if any of the computational results were novel, and to determine if the human similarity ratings between devices correlated with the computer results. We show that design software should use DC representations to produce a few focused matches which have high average weight. It should use EC representations to produce many matches some of high weight and some of low weight. Based on our human experiment, design software can use either DC or EC representations to produce novel matches. Our experiments also show that human matches correlate most strongly with a combined DC and EC representation and that their similarity reasons are more EC than DC. This suggests that designers tend to think more in EC terms than in DC terms.

Keywords: Analogy, Design, Functional Modeling, Functional Reasoning, Knowledge Representation, Repertory Grid, SME, Structure Mapping Engine.

Acknowledgements

I would like to thank David Brown for his fearless support as my advisor during my four years as a grad student.

Thank you to my wife, family, and friends for tolerating my desire to be a mad scientist on the weekends.

Thank you to my respondents for donating 1 hour of their time for the good of science.

Table of Contents

1. Introduction

Designing something is challenging, so providing computational help is important. Software systems can help the designer, or might replace the designer in some situations [Brown, 1992]. Functional reasoning is especially critical for innovative and creative design. Analogical reasoning can be used to support this kind of reasoning. However, in order for analogical reasoning to be useful, we have to describe devices in some way using a knowledge representation. This thesis is interested in quantifying what kind of analogical matches an analogical reasoning algorithm produces when the knowledge representation is varied. The results can be used to improve the reasoning capabilities of design software.

This thesis experiments with two different knowledge representations: both based on the Structure-Behavior-Function model for describing devices [Chandrasekaran and Josephson, 2000]. That work describes two different ways to represent the function of devices, Device-Centric (DC) and Environment-Centric (EC). Each may create different effects that may be advantageous for the designer. For example, EC may give the designer more "freedom." Also, the designer may decide to switch between EC and DC representation at some point during the design process in order to gain an advantage. [Chandrasekaran and Josephson, 2000]. Even though these ideas could be useful for determining when a designer should use each representation, there appears to be no research showing what the effects are of using DC verses EC representations of function.

According to Chandrasekaran & Josephson the difference between a DC and an EC representation of a device is whether or not the environment is included [2000]. For example, a DC representation of a pen's function might be "releases ink into the world" while an EC representation might be, "pen transfers ink to paper."

This thesis experiments with using both kinds of knowledge representations for analogical reasoning. Analogical reasoning involves expressing what the current situation is, looking for past situations that might apply (matching), and finally applying them to the current situation (transfer). A full study would require a system that performs all the steps in analogical reasoning, but for this thesis we take the first

step and focus only on the matching phase. We use an algorithm called SME [Falkenhainer et al., 1989]. SME was chosen because it is well tested in much research, it is claimed to have psychological backing, the software is available, and because it is suited for the problem.

Using SME we can take a pair of devices represented with a particular knowledge representation and produce a list of possible matches between them with associated weights. We measure the quantity and quality of the matches in order to measure the effect of DC versus EC representations.

We are also interested in computational support for creativity [Boden, 1994]. Analogy is often cited as a key ingredient of creativity [Goel, 1997] [Gentner et al., 2001]. As it is possible that our analogical reasoning could produce creative results, our experiment will attempt to determine whether novel matches are produced: i.e., whether DC vs. EC representations might have any effect on novelty, a key aspect of creativity [Besemer and Treffinger, 1982]. We consult a group of humans to get their judgment on the novelty of analogical matches produced by SME.

We have performed a set of experiments that that indicate where the results are coming from: i.e., the credit assignment problem. The issue is whether DC vs. EC representations, or the representation used (level of detail; ontology) should be given credit [Kitamura et al., 2004].

We show through experimentation with SME that EC produces more matches than DC, DC produces higher quality matches than EC on average, and a combined representation produces comparatively fewer matches and more lower quality matches than EC alone. These results are true even when the level of detail in the representations are varied.

In addition, from limited experiments with humans we show that they tend to rate low weighted matches as being more novel than high weighted matches and rate DC matches as being more novel than EC matches. Our human experiments also show that human matches correlate most strongly with a combined DC and EC representation and that their similarity reasons are more DC than EC.

1.1 Problem Statement

The goal of this thesis is to compare the DC and EC knowledge representations by comparing the results obtained from an analogical reasoning when using DC, EC, or combined DC and EC knowledge representations. We call this combined representation BOTH. Specifically, this thesis answers several questions, which are listed below. Our hypotheses are listed in italics after each question.

1. Which representation produces more matches?
   EC representations will produce more matches than DC representations. The BOTH representations will produce the most matches.
2. Which representation produces higher weighted matches?
   EC matches will be of lower weight than matches made using representations that are DC. BOTH matches will have the highest weights.
3. Will DC or EC representations produce more novel matches?
   EC representations will produce more novel matches than DC representations.
4. When the level of detail is varied, are the results from questions 1, 2, and 3 still true?
   Yes, the results are not sensitive to the level of detail.
5. How much do matches from each representation correlate with human matches?
   Human matches will correlate best with matches from EC representations.
6. Are human reasons for similarity more DC or EC?
   The humans' reasons will be more EC than DC.

1.2 Document Organization

The rest of this document describes the work done in this thesis and the results that were obtained. Section 2 describes the literature that is relevant. It covers functional representation, analogy, repertory grids, and creativity. Sections 3 and 4 describe how we set up the experiments, with section 3 focusing on the knowledge representation and section 4 focusing on what was needed to execute the experiment, which included test examples, details about using SME, and a test harness. Sections 5 and 6 discuss

the computation and human experiments and results. Section 7 covers an evaluation and summary of the results with respect to the original hypotheses. Finally, section 8 makes some conclusions from these results, and discusses future work.

2. Relevant Literature

This thesis investigates how changing the representation of the function of devices affects the output from an analogy-making system. It also evaluates the creativity of the output from the analogy making system. Thus, this thesis draws on four main research areas: functional representation, analogy, repertory grids, and creativity.

Section 2.1 describes functional representations that model devices and the function of those devices. That research is used to influence the representation of function used in this thesis. Section 2.2 describes what analogy is and some systems that can perform analogy. In particular, section 2.2.2 describes the particular analogy making algorithm called Structure Mapping Engine (SME), that this thesis uses for experimentation. Section 2.3 describes repertory grids, which is the technique this thesis uses to elicit similarity information from its human respondents. Finally, section 2.4 describes creativity. Information about creativity is important for understanding how to evaluate the creativity of the results produced by the analogy making system.

2.1 Functional Representation

There has been much research on how to represent devices for the purposes of reasoning about their function. An important part of the representations described in this research is representing the structure, behavior, and function of devices. Different researchers also describe various ways to represent how the device interacts with the environment and with humans.

Chandrasekaran and Josephson describe a basic ontology for structure, behavior, and function, but also make the distinction between functions that are environment-centric vs. device-centric [2000]. Section 2.1.1 goes into detail about their ideas because they are used heavily in this thesis. Section 2.1.2 describes a way to separate a device's external environment from its outer environment, thus making the distinction between the elements in the environment which are not important to the device and the ones that are. Section 2.1.3 describes the difference between a device's techno-physical environment in which the device has structure, behaviors, and functions that may or may not be a part of the human's view and the socio-cultural environment where the designer selects structure, behaviors, and functions of the device which

serve a purpose. Section 2.1.4 describes some issues researchers have faced when working with functional representations.

A related set of research is called functional modeling. Functional modeling attempts to describe a set of terms that form a common design language. Section 2.1.5 describes one such set of terms. Section 2.1.6 describes how the functional modeling research can be used in conjunction with the functional reasoning research described in sections 2.1.1 to 2.1.4.

2.1.1 Functional Ontology

Chandrasekaran and Josephson present a simple ontology for describing devices [2000]. They use this ontology to define the structure, behaviors, and functions of devices. An example of a device represented using the ontology is given in the pen example from figure 2.1. It will be used throughout this section to illustrate how the ontology works. Chandrasekaran and Josephson also explain the how functions can be represented as device-centric (DC) or environment-centric (EC) and how representations might be used by designers.

Device: Pen
Structural element 1: tip
Structural element 2: ink container
Structural relation: tip is at the end of the container
State variable 1: force on tip
State variable 2: orientation
State variable 3: location
State variable 4: ink
Causal relation 1: If the orientation is tip pointing down, the pen contacts a surface, and force is applied to the tip, ink flows out of the tip.
Mode of deployment: human orients the pen down, makes the pen contact the paper, and applies force to tip
EC function: to cause a piece of paper to have ink on it
DC function: to cause ink to flow out of the tip if the orientation is tip pointing down and force is applied to tip
Causal interaction 1: apply force from object X to object Y
Causal interaction 2: orient object X

Figure 2.1: This is a representation of a pen. It includes structural elements, state variables, a causal relation, a mode of deployment, an EC function, and a DC function. Two causal generic causal interactions are also given.

2.1.1.1 Ontology for Devices

A description of a device consists of its structure, behaviors, and functions. These concepts can be expressed using a simple ontology that consists of state variables, causal relations, and actions.

State variables describe the current properties of the device that can change. They can be any kind of value: discrete, logical, qualitative, etc. In the pen example, the pen has several state variables such as ink and orientation.

Causal relations describe how changes in one variable affect another. They can describe how the variables in one device relate, or they can describe how variables between different devices relate. If two devices are involved, the causal relation is called a causal interaction. Either kind of relation could exist in many forms. Three possible kinds of relations are simple formulas, complex algorithms, and logical expressions. In the pen example, the causal relation is a logical if-then expression, but it could also be expressed as a formula which decreases the amount of ink in the container at a certain rate.

Actions are needed to allow for reasoning about devices acting on other devices. Actions are instantiations of causal relations and causal interactions. For example, when a human uses a pen, he creates an instantiation of causal interaction 1, expressed as "apply force from pen to paper." This causal interaction describes how the pen interacts with the paper.

Views: Since the complete representation of a device is not always necessary and could be distracting, portions of a device representation can be represented in different levels of detail. Each of these variations on the representation of the device is called a view. Also, depending on the purpose of the representation and function of the device, a particular view might be more applicable.

One way to construct different views of a device is to split up the device into several components or to consider several components as the same device. In the pen example, the pen could be split up into two individual components, tip and ink container, and causal interactions could specify how they work together. However, the

view in the pen example combines the tip and the ink together so that the pen can be conveniently reasoned about as a whole.

If the function of the pen is something other than writing, then the pen might be represented in a different view. For example, if the pen is being used as a paper weight, there is no reason to represent the pen's ink container. The ink container would be left out and an additional state variable would be added to represent its weight.

A view could also be an abstraction of another view. These abstracted views may be useful for comparisons. For example, it may be difficult to directly compare a pen to a water bottle. However, if the pen is abstracted to be an ink container and the water bottle is abstracted to be a water container, it might be easier to find similarities between the pen and the water bottle since they are both containers of something. Also, abstracted views contain a mapping between the original view and the abstracted view. Thus, any information discovered between abstracted views could potentially be applied to the original view.

It is up to the designer to decide which view of the device is appropriate. If a device is too complicated, the designer might split up the device into sub devices. If a device is being used in two different ways, there may be two different views of the device. Finally, the designer might chose to abstract away certain aspects of the device for making comparisons with other devices.

Structure, Behavior, Function, and Needs: The simple ontology defined previously can be used to define a device's structure, behavior and function.

Structure is represented as state variables that have fixed values. Any structural relationships are represented as causal relations that do not change. While other causal relations may be active or inactive based on which actions have been performed, structural relations always remain stable. For modeling these, the pen example identifies two state variables as "structural elements" and one causal relation as a "structural relation."

The behavior of a device describes what a device does. However, there are several ways to express a device's behavior. A behavior can be the values of one or more state

variables at a particular instant, or the values of one or more state variables as they change over time. In the pen example a behavior might be “the ink container has 1 milliliter of ink” or “the ink in the ink container decreases.” Another way to describe behavior is to describe the behavior of something as the state of only the "output" variables, such as "the ink is decreasing" and not mention other variables in the device. In any case, the point of a behavior description is to describe what a device can do. The choice to use one behavioral description type over another depends on the context of the engineering conversation taking place.

A device's behavior is closely related to its function. The difference between a behavior and a device’s main function is that the function is intended by the designer. Section 2.1.1.2 covers functions in more detail.

Having an intended function implies that the designer has a purpose in mind for the device and the reason why the designer has this purpose is to satisfy a need. Behaviors become intended and hence a function because they serve a purpose. Also, functions ultimately exist to satisfy some need. For example, the designer of the pen could have a need, "to write my name," and the designer could assign the pen the purpose, “to write.” Any behaviors that support the pen in performing this purpose would be considered the pen’s function. The designer is satisfied because pen’s function satisfies the need, “to write my name.”

Sometimes needs are not specified in enough detail to allow a specific function to be described. For example, there is nothing in the pen representation about writing, so how does the representation able satisfy the need of writing? The answer is there is a translation step required to transform a designer's needs into a device's function.

When describing the device, Chandrasekaran and Josephson suggest a heuristic, which states that in the definition of the device’s function all translations from the need to the function are left out except the one that most closely describes the device’s function [2000]. The need "to write my name" might be translated into sub-needs such as needing ink on paper and then needing something that releases ink into the world. The function of the device is not said to be "writing." It is said to be "releasing ink." It is also possible that a designer's need, if it is specific enough, might be identical to its function. In this case no translation is required. This process is how a designer's needs

eventually get expressed in terms of function.

2.1.1.2 Environment and Device Centric Representations of Function

Functions can be described as environment centric (EC) or as device centric (DC). The notation that this section uses is summarized in table 2.1.

Notation	Description
F	a set of behavioral constraints
D	a device
W	the world or environment
Fw	an EC function
Fd	a DC function
M(D,W)	mode of deployment

Table 2.1: Functional notation

EC Representations of Function: EC representations of function describe the function in terms of the device's effect on the environment. In other words, an EC representation of function describes a device D which causes a set of behavioral constraints F to be satisfied in some world W causing an EC function Fw to occur.

The F for a Fw contains references to parts of W, but has no references to any parts of the device. In the pen example, the Fw is "to cause a piece of paper to have ink on it," F is "the paper has ink on it," and W contains a human, a pen, and paper. The Fw does not make any commitments about which device is performing the function. It only mentions how part of W must be modified in order to satisfy F. Thus, the Fw in the pen example mentions the paper, which is part of W, but not anything about the pen.

A mode of deployment, written as M(D,W), is a set of instantiated causal interactions that specify how D is embedded in W. A M(D,W) can be important to determining if a Fw is occurring because the causal interactions they instantiate could cause F to be satisfied.

There are different kinds of modes of deployment. One kind specifies the structural relationship between D and entities in W. In the pen example, specifying that the human "makes the pen contact the paper" is an example of this. Another kind of mode of deployment is a sequence of actions. A sequence of actions can be a mode of

deployment because it produces a series of relationships between the D and entities in W. The M(D,W) from the pen example is an example of this because it is made up of a series of three actions.

When a device is used with different modes of deployment, different effects may result, causing different Fw to occur. Using the M(D,W) specified in the pen example, the pen is causing the EC function “to cause a piece of paper to have ink on it” to occur. However, if the M(D,W) is "thrust the tip of the pen through the paper," the pen would perform the new function of hole punching instead. Thus, by changing the M(D,W), a device can potentially perform a different function. Having devices that can perform several functions can be desirable because such devices can reduce the number of components necessary in a design.

DC Representations of Function: In contrast to EC functions, DC functions, notated as Fd, do not mention their effect on the environment. The DC function has the assumption that desirable effects on the environment will occur so long as Fd occurs.

The F that causes Fd to occur are specified in device-centric terms. This means that the behavioral constraints in F only specify values for variables within the device or causal relations within the device. For example, the pen has a Fd of "to cause ink to flow out of the tip if the orientation is tip pointing down and force is applied to the tip." The F contains the constraints "the orientation is tip pointing down" and "force is applied to the tip." These are DC because they mention orientation and tip which are both part of D and not W. Note that the F does not mention how or what is causing the orientation to be tip pointing down or force to be applied to the tip. Presumably there is some M(D,W) that is causing it to occur, but for an Fd that M(D,W) is assumed.

2.1.1.3 Relationship Between EC and DC Device Representations

The matter of representing a device as EC or DC is a matter of convenience for the designer. One of the advantages of an EC representation is that more than one device could be used to perform the same function. This is because EC representations can be written without mentioning a specific device. For example, using the EC function from the pen example, a water bottle filled with ink could perform the same function as the pen. This shows that with EC representations, the designer has more freedom to

explore different possibilities. In contrast, DC representations are more limiting because the descriptions contain some assumptions about how it will interact with the environment. For example, because the DC function from the pen example states "force is applied to the tip" the water bottle could not perform the same DC function as the pen.

Depending on what the description is used for, an EC or DC representation might be more favorable. For example, at one point in the design process it may be useful for the designer to imagine what kinds of surfaces the pen might write on. For this, an EC representation might be more appropriate. However, for the designer who is interested in manufacturing the pen, reasoning about the surfaces external to the pen might be distracting. The designer has adequate information to manufacture the pen as long as the design states that the pen will function if ink is released from the tip. In this case, the designer might prefer a DC representation.

2.1.2 Interactions Between Devices and their External Environments

Prabhakar and Goel provide an alternate set of definitions relating to how to represent a device and its interactions with the environment [1996a][1996b]. They characterize devices that interact with the environment as low, medium, and high interaction devices (LID, MID, HID).

LIDs have a small amount of interaction with the environment that consists of a series of discrete events. Examples of LIDs include simple electronic circuits, heat exchangers, and inertia controllers.

HIDs have a high amount of interactivity with the environment. Therefore, a model of such a device needs to include a detailed model of both the device and the environment. An example of a HID is an air plane, which relies heavily on the action-reaction cycle between the plane and the air around it.

In between the two extremes are the MIDs. These devices rely on the interaction between the device and its environment, but the interactions are more limited than the HIDs. For example, an air conditioner continuously removes heat from air around it. This is a mode of interacting with the environment that is similar to what a LID would have. However, the description does not completely describe the functioning of the device because the amount of heat that has to get removed depends on the

characteristics of the environment such as the size of the room and number of items in it. Thus, to describe how the MID functions the description needs to include information about the device and the environment.

The environment in which the MID operates can be characterized as having two parts, called the external environment and the outer environment. The external environment is the physical environment outside the device. For a refrigerator, it includes the enclosure of the refrigerator as well as items outside the enclosure. The outer environment for a device is the elements in the external environment that play a role in the functioning of the device. There may be elements in the external environment that play no role in the functioning of the device. For the refrigerator, the outer environment is the food items in the refrigerator.

The difference between the external and internal environments is the kind of variables involved. The internal environment is characterized with endogenous variables such as the compression ratio of the refrigerator. The external environment is characterized by variables exogenous to the device such as the number of food items in the enclosure.

Any kind of device has an internal environment. The internal environment is particularly important for LIDs. In the internal environment, the structure the device allows it to have internal behaviors. The internal behaviors create certain output behaviors which are an abstracted form of the internal behaviors. A subset of the output behaviors can be considered the function of the device.

For MIDs it is important to describe the outer environment in addition to the inner environment. Like the inner environment, the outer environment has certain behaviors. Some of these are abstracted to be output behaviors. A subset of those output behaviors become behavior abstractions for the outer environment of the device. The inner and outer behavioral abstractions interact. A subset of those interactions become the functions of the MID.

One important distinction to make between the behaviors of the inner and outer environments is that the inner environment has intentional behavioral abstractions while the outer environment does not. The outer environment only comes into existence when part of the external environment is needed to support a behavioral

abstraction from the inner environment. This is why, for example, the outer environment for the refrigerator only contains the items in the food enclosure and not other parts of the external environment such as the shelves in the refrigerator. Since the shelves are not necessary to describe any of the inner environment behavioral abstractions of the refrigerator, they are not included in the description of the outer environment. In contrast, an inner environment behavioral abstraction, such as "remove heat from cooling liquid," exists because the designer intended it to exist. Thus, the outer environment is defined based on what the internal behavioral abstractions require.

2.1.3 Purpose and Function in Design from the Socio-Cultural to the Techno-Physical

Rosenman and Gero describe a design process that follows the following sequence:

$$P_r \rightarrow F_r \rightarrow B_r \rightarrow S \rightarrow B_a \rightarrow F_a,\{B_a,F_a,P_a\} \leftarrow\rightarrow \{B_r,F_r,P_r\}$$

Figure 2.2: P is purpose, F is function, B is behavior, S is Structure. "r" subscript means " required", "a" subscript means "actual." The symbol "→" is a convert step, and the "←→" symbol is a compare step [Vermaas, 2002].

The first two steps (Pr → Fr, and Fr → Br) are called problem formulation and involve the processes of translating required purposes into required functions and required functions into required behaviors. Behaviors are then converted to structure. Once the structure exists, the actual device has to be analyzed such that "Structure exhibits Behavior effects Function enables Purpose." Finally, the behavior, function, and purpose of the actual device are compared with the required behavior, function, and purpose. If there is a discrepancy, the design process begins again with a reformulation.

Rosenman and Gero distinguish which of these processes occur in the socio-cultural and techno-physical environments [Rosenman and Gero 1998]. In the socio-cultural environment, the human creates purposes and evaluates utility of the function, behavior, and structure with respect to the purposes. By doing so, the human creates a view of the device that relates to desired intentions.

In the techno-physical environment, the device has structure, behaviors and functions, which interact with the natural environment. Part of the techno-physical environment consists of structure, behaviors, and functions that the human intended.

However, the device also interacts with the natural environment which behaves according to the physical laws. This may cause the device to have unintended behaviors. Also, the techno-physical environment may have irrelevant structures which are not part of the human's view of the device.

The interaction between socio-cultural and techno-physical environments is such that if a device is taken out of a socio-cultural environment and put in another, it will have the same techno-physical environment, but different purposes and functions. The distinction is useful because it allows the device to be represented in different views based on the designer and use of the device, without changing the entire representation. Allowing a device to have different views may allow for new uses of a particular device to be uncovered.

2.1.4 Functional Reasoning in Design

Functional reasoning is an important concept in a widely accepted design methodology described by Pahl and Beitz [2003]. Using this methodology the designer specifies the function for the entire product, splits the function into sub functions, looks up elements that can perform the functions, and composes a solution based on the elements.

Despite the fact that functional reasoning is a major part of the design process, current CAD systems typically only support geometric modeling. To further support designers, future systems should support the entire process including functional reasoning. These systems should do this because functional reasoning has many advantages, including helping to determine a products basic characteristics and helping to decompose the design problem. Also, products that have problems with their main functions do not sell very well. Umeda and Tomiyama provide an overview of the various issues involved in defining function and implementing functional reasoning in CAD systems [1997].

The definition of function can be different for each researcher. Researchers agree that function is related to behavior, but they disagree about the definition of function in two ways. First, a function could contain the designer’s intention. When it is included, the function includes the reasons why the behaviors are required and the

intentions can be represented explicitly or inferred. If it is not included, the function is just an abstracted behavior. There may be some advantages to representing the designer's intention in the definition of function since the intentions can be used to support design activities such as verification, reuse, or explaining results.

The second issue in the definition of function is the behavioral representation. Some possible approaches include either state transitions, bond graphs, functions, and behavior structure (FBS) modeling. These approaches differ because of their application domain. A particular approach is good for some tasks but struggles at others. For example, bond graphs are appropriate for power systems design, but it is hard to use bond graphs to represent devices that do not transform anything. Thus, researchers are still investigating the question of when to use a particular behavioral representation.

While implementing advanced CAD systems that perform functional reasoning, researches have learned some lessons. Representing function helps organize designs for reuse. The ability to verify designs early on using simulation is critical. In order to represent functions, a designer must be experienced. Functional CAD systems should be able to deal with quantitative attributes and geometry to make it easier to bridge the gap between existing design systems.

There are two additional issues that future CAD system designers must face in order to make CAD systems that go beyond verification of existing designs and configuration designs.

The first issue is that designers need to be able to design from the view point of structure, behavior, and function. Since a certain function might have very different behaviors and structural hierarchies, future CAD systems must figure out how to make representations consistent and useful.

The second issue is a top-down versus bottom-up issue. Since systems can be designed starting with the structure and then finishing with function and vice versa, a functional reasoning tool should be able to combine the two kinds of reasoning.

2.1.5 Functional Basis

The functional basis [Stone and Wood, 1999] is a common design language that can be used for functional modeling. It consists of two main parts: functions and flows. A

function is a description of an operation performed by a device. The reader should note that this definition of function is different from other definitions described in this section. For example, the work described in section 2.1.1 would classify these as behaviors not functions. A flow is the change in material or energy caused by a function. A flow is the recipient of the function's operation. Concepts using the functional basis are expressed as verb-object pairs, where functions are the verbs and flows are the objects.

Flows represent the quantities that are input and output by functions. For example, the function *convert* could take the flow *human force* as input and output *mechanical force.* Flows are broken down into three classes: material, signal, and energy. Signal flows are actually made of material or energy, but they are given a special classification in the functional basis. Each class has basic and sub-basic flows such as the basic flow *human* or *mechanical* and the sub-basic flows *human hand*, or *mechanical force*.

Flows can be expressed in three ways depending on how specific the description needs to be. The most general description is just the class expressed as *human* or *signal*. A more specific flow is the basic description + class pair such as *human energy*, or an even more specific flow is the sub-basic description + class pair such as *human force*. Depending on the customer needs, the designer may use more general flows to allow a more general description and use variants, or specific flows to give a more detailed, concrete design. The functional basis also provides clear written definitions for each flow.

Functions are defined in eight classes with basic functions in each. The functional basis provides clear definitions for each basic function as well as lists the synonyms that might be used to represent that function. Some example functions include *import*, *export*, *transmit*, *couple*, *display*, *rotate*, and *change*.

The authors of the functional basis suggest a way to apply the functional basis to designing. They suggest that the first step is to figure out a black box model of the product and to identify the flows in and out of the model. Then, the designer creates function chains for each input flow, envisioning how the flow moves through the device. The designer expresses each change in the flow as a sub-function using the

functional basis vocabulary. The next step is to order the function chains by time. There may be sequential or parallel function chains in the system. The final step is to aggregate all the flows together connecting them as necessary by possibly adding new sub-functions.

There are several advantages to using the functional basis. First, the functional basis can help designers to make a product architecture more modular earlier in the design process. This is done by grouping sub functions together.

Second, the functional basis allows functional models to be expressed in a consistent way. The functional basis allows functional models to be consistent because each model uses the same set of terms and because each term has a clear definition.

Third, functional models can be stored in a corporate body of design knowledge. Designers can use the stored models to find products of with similar functions, or products that are directly usable.

Fourth, functional models can aid in creative concept generation because they provide a way to represent abstract or incomplete information, and because they can help decompose a problem into sub functions.

Finally, functional models can reduce the guesswork involved in creating metrics for a certain product. Instead of defining a new set of metrics for each product, metrics can be defined over a range of products. One type of metric could be a high-level physical model of a product's technical progress. Other types of metrics could measure product benchmarks and product quality.

2.1.6 Relationship Between Functional Modeling and Functional Reasoning

Sections 2.1.1 through 2.1.4 describe different parts of functional reasoning (FR) research. Section 2.1.5 describes one kind of functional modeling (FM) research. These two research areas are in fact related and are complementary to each other [Chandrasekaran, 2005].

The definition of function in FR and FM are similar, but not the same. Both research areas agree that a function is what the device does and that the vocabulary for functions is the same as behaviors. However, FR makes the further distinction that a function is only the set of behaviors that are desired. Therefore, according to the FR

terminology, FM is doing something more like “behavior modeling” than “functional modeling” because for FM all the behaviors can be considered functions.

Another difference between FR and FM research is that FM research is not driven by a need to experiment with automated reasoning, while FR is. FR research worries about the fine details of the representational aspects of devices so that it can use functional representations for automated reasoning. Having this automation can allow a system to determine, for example, if a device actually achieves the desired function. FM is not as formal as FR, one only needs an intuitive understanding of what the terms mean in order to use the system. However, ontology development is in fact a challenging process that requires extensive experimentation.

Therefore, FM research could benefit from utilizing FR research. Doing so would allow FM researchers to make their primitives more precise. One way that FR research could make FM research more precise is by making the distinction between DC and EC representations. This distinction is currently not made in the FM research.

Researchers from FR and FM communities could benefit from each other’s work because FM refines the general ontologies that FR defines. The behavior primitives described in FM research can become a content theory for FR. This can be very useful because FR has no specific primitives for properties, behaviors, and functions in specific domains. Another way to describe this is FM specifies what variables exist in the domain, and FR specifies what types of variables might exist in the domain. FR describes what kinds of objects are involved in making devices and FM refines the kinds of behaviors that can exist for certain subclasses of devices. Having a theory that uses both FR and FM would be applicable to wider variety of domains than a theory that just encompasses only FR or FM.

2.1.7 Summary

Functional representation research forms the theory used in this thesis to design the knowledge representation and test examples. There are several views about how to describe devices. These views help to influence the knowledge representation used in this thesis and show why functional representation is important for computer-based systems that support design.

The functional ontology described in section 2.1.1 shows how the represent the structure, behavior, and function of devices using a simple ontology and, in particular, it describes how to represent devices in DC and EC ways. The DC vs. EC difference is the main variable varied in the experiments for this thesis.

The environment can be split into an outer environment and external environment. For some devices it is necessary to represent the environment in order to accurately describe its function. This means that a precise understanding of how to model the environment is important.

Interaction between the device and the environment can also be viewed as an interaction between the techno-physical and socio-cultural environments. This distinction allows the device to have many kinds of structure, behaviors, and functions in the techno-physical environment, but only a subset which are relevant to the human in the socio-cultural environment. This indicates that functions exist to satisfy a designer's purpose and the designer might have different purposes for the same device. Thus, the device remains largely the same, but the designer will assign it different functions depending on the particular socio-cultural environment.

Lastly, Umeda and Tomiyama say that the behavioral representation is important for systems that perform functional reasoning and that there are several competing approaches [1997]. Thus, the results from this thesis will be useful because they help define which kind of behavioral representation is useful in which circumstances.

An area of research related to functional representation is functional modeling. The functional basis is a language for functional modeling that provides a set of domain specific terms for describing flows and functions. In this thesis, these terms are used in conjunction with functional representation theories to create accurate test examples.

2.2 Analogy

Analogical reasoning is, in fact, a fundamental reasoning process that people use all the time in everyday life [Gentner et al., 2001, p. 499-537]. It is also a particularly important process for producing creative designs [Pahl and Beitz, 2003] and for inventing [Wolverton and Hayes-Roth, 1995]. Because of the importance of analogical reasoning, researchers have developed a good understanding of analogy making and several analogical systems have been built.

Section 2.2.1 reviews analogy in general. Section 2.2.2 describes an algorithm for performing analogical matching that will be of particular use in this thesis. Finally, section 2.2.3 and 2.2.4 describe selected systems that perform design by analogy. Overall, this section provides an overview of analogy and some analogy systems that can perform it.

2.2.1 Analogy in General

Analogy is "the ability to identify patterns, to identify recurrences of those patterns despite variation in the elements that compose them" [Gentner et al., 2001, p. 2]. In particular, analogy is the ability to think about relational patterns. For example, if two circles are compared to two possible analogs, two squares or a square and a triangle, the best analog is the two squares because they both share the relationship: sameness of shape. The importance of this analogy is that the analogy between the two circles and two squares relies on a common relationship, not their physical appearance. In order to make this analogy, a person needs to represent and reason about the relationship between the objects [Gentner et al., 2001, p. 2].

The analogy making process can be broken down into a series of steps in order to make a mapping between two domains, called the source and target. The target is the new description that must be matched with a known source. The source is sometimes called the base. First, the analogy system must access relevant source analogs from long term memory. Second, parts of the source are mapped to the target. Third, analogical inferences are made between the source and target to fill in any missing knowledge in the target. Finally, learning occurs as the new analog is incorporated into the analogy system's memory [Gentner et al., 2001, p. 9].

There are three issues that computer-based analogy systems face. First, in order to make more complicated analogies, more complex representations are necessary. Thus, any computational system must be able to build and manipulate complex representations. The second issue analogy systems face is the "binding problem." This problem involves identifying the roles for a particular piece of knowledge. A third issue is the need for representations that are dynamic enough to allow a reasoner to change the source and target representations during the reasoning process. Since the

reasoner might be trying to find analogies between different domains which have very different representations, the representations might need to change significantly in order for them to be compared.

There have been three approaches for implementing analogy systems. One is based on using methods such as logic, planning, and search, and another is based on connectionist methods that use nodes, weights, and spreading activation in a network [French, 2002]. The final approach is a hybrid of the first two approaches.

Symbolic methods do well at dealing with the first two issues analogy systems face because they have explicit symbols to represent the analogies and relationships between elements in the analogies. The two issues are more of a challenge for connectionist approaches, which uses activations over a neural substrate to represent symbols instead of using explicit representations. However, connectionist approaches have the advantage that they provide a natural internal measure of similarity [French, 2002]. Both kinds of analogy systems have problems with the third issue.

An example of a symbolic method for analogy making is an algorithm called Structure Mapping Engine (SME). SME makes analogies based on the structural similarity between two domains. Thus, analogs are mapped based on the relationships rather than on the attributes of the source and target. The algorithm also uses the systematicity principle which states that larger, more coherent mappings are preferred over individual mappings thus allowing it to build complex analogies.

An example of a connectionist method for analogy making is ACME [Holyoak and Thagard, 1989]. ACME uses an architecture based on the parallel activation of nodes in a neural network-like structure. It frames the problem as a constraint satisfaction problem. The system represents the pairings between the source and target as links between nodes in a neural network [French, 2002]. When the system is presented with source and target representations, certain links get deactivated and the most active hypothesis becomes the best analogy.

An example of a hybrid approach is a model like AMBR [Kokinov and Petrov, 1988]. AMBR has symbolic methods that encode declarative and procedural knowledge. AMBR has a connectionist part that computes the activation level of a particular reasoner in the system. When a reasoner is more activated its actions are

more relevant. Using this connectionist model, AMBR is able to process an analogy all at once without a preset order of steps.

2.2.2 SME

As mentioned in the previous section, the Structure Mapping Engine (SME) is a kind of analogy making system. Since this thesis will be using SME, this section goes into more detail about how SME works. It describes the overall algorithm, and pays particular attention to how changing the SME parameters affects its output. For a more complete description of the algorithm see [Falkenhainer et. al., 1989].

2.2.2.1 Structure Mapping Theory

SME is an implementation of the psychological theories of Gertner [1983]. It is an analogical matching algorithm that produces mappings between parts of source and target representations. As of 1990 there were over 40 projects used it [Falkenhainer et. al., 2005]. In a more recent review, French said Structure Mapping Theory is "unquestionably the most influential work to date on the modeling of analogy-making" [2002].

SME is useful because it ignores surface features and finds matches between potentially very different devices if they have the same representational structure. For example, SME could determine that a pen is like a sponge because both are involved in dispensing liquid, even though they accomplish it very differently.

Structure Mapping Theory is based on the systematicity principle, which states that more connected knowledge is preferred over independent facts. Therefore, SME should ignore isolated source-target mappings unless they are part of a bigger structure. SME should map objects that are related to knowledge already mapped.

Structure Mapping Theory also requires that mappings be one-to-one, which means that no part of the source description can map to more than one item in the target and no part of the target description can map to more than one part of the source. In addition, structure mapping theory requires that if a match maps S to T then the arguments of S and T must also be mapped. If both these conditions are met, the mapping is said to be structurally consistent.

2.2.2.2 SME Algorithm

SME takes two descriptions called the source and target, and maps knowledge from the source into the target. SME calls each description a *dgroup*. Dgroups contain a list of *entities* and *predicates*. Entities represent the objects or concepts in a description such as an inputgear or a switch. Predicates are one of three types and are a general way to express knowledge for SME. *Relation* predicates contain multiple arguments which can be other predicates or entities. An example relation is: (transmit (what from to)). This relation has a functor "transmit" and takes three arguments: "what," "from," and "to." *Attribute* predicates are the properties of an entity. An example of an attribute is (red gear) which means that gear has the attribute red. Finally, *function* predicates map an entity into another entity or constant. An example of a function is (joules powersource) which maps the entity powersource onto the numerical quantity joules. Functions and attributes have different meanings and consequently SME processes them differently. For example in SME's true analogy rule set, attributes differ from functions because they cannot match unless there is a higher order match between them. The difference between attributes and functions will be explained further in this section's examples.

All predicates have four parameters. They have a functor, which identifies it and a type, which is either relation, attribute, or function. The other two parameters are for determining how to process the arguments in the SME algorithm. If the arguments have to be matched in order, commutative is false. If the predicate can take any number of arguments, N-ary is false. An example of a predicate definition is:
(sme:defPredicate behavior-set (predicate) relation :n-ary? t :commutative? t)
The predicate's functor is "behavior-set," its type is "relation," and its n-ary and commutative parameters are both set to true. The "(predicate)" part of the definition specifies that there will be one or more predicates inside an instantiation of behavior-set.

The first step of the algorithm is to create a set of *match hypotheses* between source and target dgroups. A match hypothesis represents a possible mapping between any part of the source and the target. This is controlled by a set of match rules. By changing the match rules, one can change the type of reasoning SME does. For

example, one set of match rules may perform a kind of analogy called “literal similarity” and another performs a kind analogy called “true-analogy.” These rules are not the place where domain dependent information is added, but rather where the analogy process is tweaked depending on the type of cognitive function the user is trying to emulate.

There are two types of match rules: filter rules and intern rules. Intern rules only use the arguments of the expressions in the match hypotheses that the filter rules identify. This makes the processing more efficient by constraining the number of match hypotheses that are generated. At the same time, it also helps to build up the structural consistencies that are needed later on in the algorithm. An example of a filter rule from the true-analogy rule set creates match hypotheses between predicates that have the same functor. The true-analogy rule set has an intern rule that iterates over the arguments of any match hypotheses, creating more match hypotheses if the arguments are entities or functions, or if the arguments are attributes and have the same functor.

In order to illustrate how the match rules produce match hypotheses consider these two predicates:

transmit torque inputgear secondgear (p1)
transmit signal switch div10 (p2)

The filter match rule generates a match between p1 and p2 because they share the same functor, “transmit.” The intern rules then produce three more match hypotheses: torque to signal, inputgear to switch, and secondgear to div10. The intern rules created these match hypotheses because all the arguments were entities.

If the arguments were functions or attributes instead of entities, the predicates would be expressed as:

transmit torque (inputgear gear) (secondgear gear) (p3)
transmit signal (switch circuit) (div10 circuit) (p4)

These additional predicates make inputgear, secondgear, switch, and div10 functions or attributes depending on the value defined in the language input file. The representation also contains additional entities for gear and circuit.

Depending on what type inputgear, secondgear, switch, and div10 are, their meanings change. As attributes, each one is a property of the gear or circuit. For example, the gear has two attributes, inputgear and secondgear. The circuit has two attributes, switch and circuit. As functions inputgear, secondgear, switch, and div10 become quantities of the gear and circuit. In this example, the functions inputgear and secondgear now map to the numerical quantities "torque from inputgear" and "torque from secondgear," For the circuit the quantities map to logical quanitity "switch engaged" and the numerical quantity "current count on the divide by 10 counter."

SME processes these differently. It does not allow attributes to match unless they part of a higher order relation, but it does allow functions to match, even if they are not part of a higher order relation. It allows functions to match because they indirectly refer to entities and thus should be treated like relations that involve to entities. However, as section 2.2.2.3 shows, the intern rules assign lower weights to matches between functions than matches between relations. The reason why SME does not match attributes is because it is trying to create connected knowledge based on relationships and thus satisfy the systematicity principle. For example, if both a clock and a car have inputgear attributes SME will not mark them as similar. If it did, it would be making a match between the clock and car based on their appearance not on the relationships between them.

When the additional predicates in p3 and p4 are functions, the results from matching p3 and p4 are similar to the results from p1 and p2 except there is an additional match between gear and circuit and the values for the match hypotheses between (inputgear gear) and (switch circuit), and (secondgear gear) and (div10 circuit), are lower. Section 2.2.2.3 describes the reason for this in more detail.

If the inputgear, secondgear, switch, and div10 are attributes instead of entities, SME does not find matches between any of the attributes. It only finds matches between the transmit predicates and between torque and signal. Additionally, the structural evaluation scores for the remaining two matches decreases. In order to get the two predicates to match, p3 would need to be replaced by p5. P5 is shown below.

transmit torque (inputgear gear) (div10 gear) (p5)

Since the true-analogy rule set identifies that the div10 attributes are the same between p5 and p4 and because the div10 attributes are both part of the higher relation match between torque and signal SME makes a match between (div10 gear) and (div10 circuit) which leads to a match between gear and circuit.

Being part of a higher order match is a requirement only for attributes. For example, if (div10 gear) and (div10 circuit) are not part of a higher order match, SME does not create a match hypothesis between match them. However, if div10 is a function or relation SME does create a match.

2.2.2.3 Structural Evaluation Score

Once the match hypotheses are generated, SME needs to compute an evaluation score for each match hypothesis. SME does this by using a set of intern match rules to calculate positive and negative evidence for each match. Multiple amounts of evidence are correlated using Dempster's rule [Shafer, 1978] resulting in positive and negative belief values between 0 and 1. The match rules assign different values for matches involving functions and relations. These values are programmable, however some default values that can be used to enforce systematicity principle are described in [Falkenhainer et. al., 1989].

These rules are:

1. If the source and target are not functions and have the same order the match gets +0.3 evidence. If the orders are within 1 of each other, the match gets +0.2 evidence and -0.05 evidence.
2. If the source and target have the same functor, the match gets 0.2 evidence if the source is a function, and 0.5 if the source is a relation.
3. If the arguments might match, the match gets +0.4 evidence. The arguments might match if all the pairs of arguments between the source and target are entities, if the arguments have the same functors, or it is never the case that the target is an entity but the source is not.
4. If the predicate type matches, but the elements in the predicate do not match, then the match gets -0.8 evidence.
5. If the source and target expressions are part of the a matching higher order match, add 0.8 of the evidence for the higher order match.

In the example match between p1 and p2, SME gives the match between the transmit relations a positive evidence value of 0.7900 and the others get values of 0.6320. The transmit relation receives the evidence value of 0.7900 because it gains evidence from rules 1, 3, and 2. The other matches get a value of 0.6320 because 0.8 of the evidence from the transmit is propagated to these matches because of rule 5.

For predicates p3 and p4, SME assigns less evidence because the arguments of the transmit relations are functions. The transmit relation gets positive evidence of 0.65 because rule 3 no longer adds evidence. The match between (input gear) and (switch circuit) becomes 0.7120. This match gets 0.4 evidence because of rule 3, and 0.52 evidence propagated from the transmit relation because of rule 5.

When the predicates in p3 and p4 are attributes, rule 4 adds -0.8 evidence to the transmit match because though the functors of the transmit relation match, the arguments do not have the potential to match and the arguments are not functions.

To summarize, the intern match rules compute a structural evaluation score for each match hypothesis. These rules enforce the systematicity principle. Rule 5 provides trickle-down evidence in order to strengthen matches that are involved in higher order relations. Rules 1, 3 and 4 add or subtract support for relations that could have matching arguments. Rule 2 adds support for when the functors match thereby adding support for matches that emphasize relationships.

The rules also enforce the difference between attributes, functions, and relations. For example, they have checks which give less evidence for functions than relations. Attributes are not specifically dealt with by the intern match rules, but SME's filter rules ensure that they will only be considered for these rules if they are part of a higher order relation and rule 2 ensures that attributes will only match if they have identical functors.

2.2.2.4 Gmap Creation

The rest of the SME algorithm is involved in creating maximally consistent sets of match hypotheses. These sets of match hypotheses are called *gmaps*. SME must ensure that any gmaps that it creates are structurally consistent. This means that they are one-to-one, such that no source maps to multiple targets and no target maps to multiple sources. It also means that they must have support, which means that if a

match hypothesis is in the gmap, then so are the match hypothesis that involve the source and target items.

The gmap creation process follows two steps. First, SME computes some information about each match hypothesis. This includes entity mappings, what other match hypotheses it conflicts with, and what other match hypotheses it is structurally inconsistent with.

SME then uses this information to merge match hypotheses using a greedy algorithm and the structural evaluation score. It merges the match hypotheses into maximally structurally consistent connected graphs of match hypotheses. Then it combines gmaps that have overlapping structure if they are structurally consistent. Finally, it combines independent gmaps together while maintaining structural consistency.

Comparing a source to a target dgroup may produce one or more gmaps. The weight for each gmap is the sum of all the positive evidence values for all the match hypotheses involved in the gmap. For example, if a source containing p1 and p6 below, is compared to a target containing p2, SME will generate two gmaps. Both gmaps have a weight of 2.9186.

Source:

transmit torque inputgear secondgear (p1)

transmit torque secondgear thirdgear (p6)

Target:

transmit signal switch div10 (p2)

```
Gmap #1:
(TORQUE SIGNAL)
(INPUTGEAR SWITCH)
(SECONDGEAR DIV10)
 (*TRANSMIT-TORQUE-INPUTGEAR-SECONDGEAR
  *TRANSMIT-SIGNAL-SWITCH-DIV10)

Gmap #2:
(TORQUE SIGNAL)
(SECONDGEAR SWITCH)
(THIRDGEAR DIV10)
(*TRANSMIT-TORQUE-SECONDGEAR-THIRDGEAR
 *TRANSMIT-SIGNAL-SWITCH-DIV10)
```

Figure 2.3: Gmaps resulting from comparing a source containing a p1 and p6 and a target containing p2.

The gmaps in figure 2.3 show pairs of predicates or entities that match. For example in gmap #1, the entities torque and signal match and the behaviors transmit torque inputgear secondgear and transmit signal switch div10 match. Gmap #1 represents combining p1 and p2. Gmap #2 represents combining p1 and p6. Although p2 is compatible with both p1 and p6, the one-to-one mapping constraint enforces that both mappings cannot be in the same gmap. Therefore, SME produces two independent gmaps. In addition, combining the two gmaps together would make the entity mappings between thirdgear and div10 conflict with the entity mapping between secondgear and div10.

2.2.3 KDSA

Wolverton and Hayes-Roth describe a system called $KDSA_{ID}$, which is designed to find semantically distant, innovative analogies between devices [1995]. It is based on three observations of how inventors use analogies. First, inventors draw analogies from an unpredictable number of domains that can be a very different from each other.

Second, inventors use concepts that are unusual or unexpected to find more analogies. Thus, concepts that are as different as possible from the target concept while still being useful are the best for innovative analogies. Useful means that only the features that are necessary for the device to function are included and any extraneous features are mismatched as much as possible. Surface similarity is not good for innovative designs.

Third, inventors can stumble across a solution while working on the design problem. They proceed using a conscious or unconscious search of memory. Inventors might find a phenomenon and search for a problem to apply it to. They could also start with a problem and search for a solution or start with a solution and search for a problem.

Wolverton and Hayes-Roth developed an algorithm called KDSA and the knowledge that applies it to design, called $KDSA_{ID}$. The KDSA algorithm retrieves semantically distant analogues and then uses heuristics defined by $KDSA_{ID}$ to guide KDSA to useful analogues between devices.

KDSA represents the world as a single semantic network which has nodes that are associated with links. To retrieve a concept, the target concept nodes, and possibly nodes representing characteristics of the solution, are activated.

KSDA has several steps. First, the graph matcher maps concepts to the target. Second, mapping evaluation evaluates the map using a task specific similarity metric. Third, search control uses heuristics to focus the spreading activation search. KDSA proceeds to search for an analogue until it finds one that exceeds a desired set of thresholds.

A distinguishing characteristic of this approach is that the mapping evaluation step provides feedback to the search control step and vise versa through changes to the semantic network. Other analogy algorithms serialize retrieval and mapping as independent processes.

$KDSA_{ID}$ adds heuristics for the map evaluation and search control stages of KDSA. The heuristics for map evaluation are set up to make source and target devices have similar functions but different behaviors. This ensures that both devices will be able to perform the same function, but that they will be more novel in the behaviors used to accomplish the function. Second, the map evaluation makes sure the source is adaptable to the target so that if a possible match is found the match can be mapped back to the target. Map evaluation makes sure that the analogies are not so different that they are useless.

To get these effects, the map evaluation component makes decisions based on two measures: isomorphism and semantic distance. Isomorphism is the percentage of

nodes and links that match between the target and source. Semantic distance is the average path distance between mapped nodes. $KDSA_{ID}$ defines different similarity metrics based on these measures. It defines thresholds on these values so that it can stop the search when a suitable analogy is found. For example, one of the conditions requires that the analogy must have high isomorphism. Another requires that the distance measure between the structure of the devices in the source and target is high. These measures are set to encourage $KDSA_{ID}$ to find innovative designs.

$KDSA_{ID}$ also defines some heuristics for search control. The heuristics are set to increase likelihood of future mappings working and to reduce amount of search necessary to do it. The "activate promising concept" heuristic strengthens the activation levels of parts of the concepts that are close to meeting the thresholds. "Prune unpromising concept" clears activation for unpromising concepts and makes it so they cannot be activated. "Cross-domain bridge" utilizes known abstractions to move analogues out of the same domain. "Modify retrieval condition" makes is so that devices are only retrieved if the representation of its behaviors are highly activated.

In order to make KDSA useful, the heuristics and thresholds must be set so that "flaky", useless analogies are eliminated, but yet KDSA is still allowed to find novel, surprising analogies. For example, Wolverton and Hayes-Roth mention KDSA found an innovative analogy between a rock crusher and an irrigation system that suggested that the irrigation system should transport water on a conveyor belt [1995]. When the researchers added an "adaptability requirement" to one of their heuristics, the system no longer found that analogy. Thus, adding too many constraints to an analogy system could eliminate potential innovative analogies.

2.2.4 Qian and Gero

Qian and Gero describe an analogy making system called DESSUA that utilizes a knowledge representation consisting of qualitative causal relations and generalized design knowledge of devices to perform creative design [1992][1996]. The knowledge is put into three categories: structure, behavior, and function. The analogy system has several parts: a concept retriever, analogy retriever, an analogy elaboration step, and an evaluation step.

The concept retriever searches for a source design based on the conceptual design name and design requirements. The design requirements are expressed with the same structure, behavior, function model as the devices in the design library. The concept retriever uses the conceptual design to find an existing design and uses that design to generate analogical retrieval clues in the form of a target concept.

The analogy retriever retrieves analogous designs that have the same function or behavior as the design in the target concept. It may also retrieve designs based on the design requirements.

The elaboration step identifies correspondences between the source and target. If two functions match, then the behavior, structure, and external effects are mapped. If a behavior matches, only the structure can be mapped. Thus, matching can occur between two function variables, two behavior variables, or two behavior graphs if they are the same at some abstract level. Variables that represent structure and exogenous variables can get mapped only if they have the same associated functions or behaviors.

The evaluation step requires a human to comment on the system's output. This is needed because the design variables introduced from the source to the target may not have associated domain knowledge. Therefore the system cannot evaluate them.

Qian and Gero show that the system is capable of designing devices by analogy. For example, it designed a buzzer based on an analogy with a blinking cursor. It also designed a new kind of folding door based on an analogy with a curtain.

2.2.5 Summary

There are many different analogy systems that have been built. Some of them use symbolic representations like SME and DESSUA, and others like KSDA use a network like structure to make analogies.

SME is effective at finding analogies based on relationships. This means that SME could possibly find analogies that come from different domains. Such analogies have the potential to be seen as creative. Since this thesis seeks to measure if creative analogies can be produced, this is an important property of SME. Also, the format of the SME input works off of symbols. This is important because the knowledge

representation this thesis uses is based on the theories described in section 2.1, which largely discuss symbolic representations of functions that include objects such as "behaviors" and "functions."

A notable feature of KSDA is that it uses thresholds to control when a suitable analogy was found. The researchers who developed KSDA found that if they over constrain the system, it does not produce some interesting analogies. This lesson could be relevant to this thesis since one representation might be more constraining than another. One representation may produce too many analogies that are not all useful or one may produce too few. As KSDA has shown, the parameters of the analogy making system need to be tuned to produce enough useful analogies without producing too many useless ones. It is possible that the representation type used could be one of these parameters.

Lastly, an important lesson for this thesis from the work on DESSUA is that the human needs to be involved in the evaluation especially when the products lack represented domain knowledge. This suggests that humans might need to be involved in some part of the evaluation process used in this thesis.

2.3 Repertory Grids

One way to measure how computer based analogical reasoning systems, such as the ones described in section 2.2, perform is to compare the results to what people produce, since people also perform analogical reasoning. In order to make this comparison, this thesis requires a technique such as repertory grids to extract measures of similarity from human experts.

The repertory grid technique is well suited for this purpose. Since computational analogical reasoning systems can produce the same kind of similarity measure that the repertory grid analysis produces, repertory grids can be used to compare human results to the computer's results. Additionally, there is a software package that simplifies collecting and analyzing repertory grids.

2.3.1 Repertory Grid Technique

The repertory grid is a technique for eliciting knowledge about the way an expert categorizes the world and reasons about it. It allows a knowledge engineer to record

the expert's view of a particular problem and to get the expert thinking about the problem [Hart, 1986].

A repertory grid involves two concepts: elements and constructs. Elements are the items in the world that the expert is trying to categorize. Constructs are bi-polar scales that the expert uses to rate each element. The scale is a numeric scale, such as 1 to 5, and the expert names each pole. For example, if an expert were describing a set of people elements. The expert might create a construct where 1 is "short" and 5 is "tall" and then rate each person on a height scale of 1 to 5.

Since each expert may provide a wide array of constructs, it is important to note that the ratings that a particular expert gives are useful for comparison purposes only. It does not make sense to compare two experts' ratings, even if they have exactly the same constructs.

Also, the ratings are only relative. For example, if an expert gives Brian a rating of 4 on the height scale, and gives Sue a rating of 2, it does not mean that Brian is twice as tall as Sue, it just means that Brian is taller than Sue. Figure 2.4 is an example of a grid.

Figure 2.4: Example of a repertory grid for people. Elements are on the bottom and constructs are on either side. This figure was made using software by Shaw and Gaines [2005].

Eliciting a grid is an iterative process, between a knowledge engineer and the expert. The process ends when the expert is satisfied that the grid accurately reflects his or her views. The expert could just fill out the grid directly by naming all the elements and constructs. However, this is often to difficult for an expert, so the elicitation process is usually an iterative process, where the knowledge engineer asks the expert to evaluate triads of elements at a time. For example, in eliciting the above grid, the knowledge engineer might ask how Sue, Bill, and Charlie are in some way

different. The expert's response is that Charlie likes sports and Sue and Bill do not and the expert rates them as such. The advantage of using triads is that a triad of elements is the minimum the expert needs to evaluate in order to identify one difference and one similarity. The small number of elements is easy for the expert to evaluate. By comparing enough sets of triads, the expert eventually fills in the entire grid. The expert is allowed to add elements, or change constructs at any time during the elicitation process.

Once the grid is elicited, the grid can then be analyzed by a clustering technique. This clustering technique involves two steps. First, it computes a percent similarity measure between each element and construct. Then, the clustering technique orders the elements and constructs into a "focused" grid that helps to show the expert which elements are most similar.

Figure 2.5 shows an example of a focused grid, which is based on the percent similarity measures from the cluster analysis. The elements that are most similar are next to each other. For example, Kelly is most like Sue, and Bill is most like Charlie and Brian. The lines above the elements represent the percent similar measure between elements. The lines show Kelly and Sue are about 80% similar. When the lines connect it means that all elements in the cluster are at that level of similarity. Thus, John, Brian, Bill, and Charlie are all about 70% and all the elements are about 45% similar. A similar arrangement is made for the constructs and shown with the lines on the right.

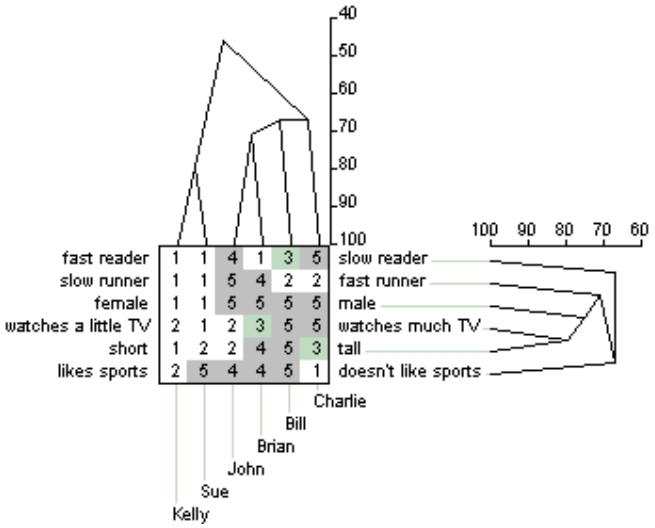


Figure 2.5: A focused grid for people. The elements are on the bottom and the constructs are on either side. The lines above the elements and beside the constructs show the clusters on a scale of percent similar. This figure was made using software by Shaw and Gaines [2005].

The cluster analysis computes a measure of difference between each element or construct and then computes their percent similarity. Difference is measured by the sum of the absolute differences in the ratings. Thus, the ratings for John are: 4 5 5 2 2 4 and for Brian the ratings are: 1 4 5 3 4 4. The differences between ratings are 3 + 1 + 0 + 1 + 2 + 0 = 7. To compute the difference measure between two elements or constructs the formula is:

$(-100D_{ij} / (m * n)) + 100$

Where D_{ij} is the difference between element or construct i and j, m is the maximum difference between elements, and n is the total number of elements or constructs in the grid. Therefore, the percent similarity between John and Brian is 71%.

Each construct undergoes an extra step in analysis that does not occur for the elements. A construct may give different similarity measures when depending on which pole is the low pole and which is the high. When a reversed construct gives more similarity, the cluster analysis uses it instead of the original.

There are several advantages of the grid. First, the grid can be analyzed using techniques such as clustering and then the results can be compared to grids from other experts. Second, the grid makes the expert think carefully about the problem, thereby clarifying the expert's views and explicitly representing their implicit knowledge.

2.3.2 Repertory Grid Software

Many tools exist to make the repertory grid elicitation and analysis easy to do. One tool is Rep IV [Shaw and Gaines, 2005] which is a commercial tool that is free for academic use. This tool helps elicit a grid from an expert and performs all the repertory grid cluster analysis.

The elicitation tool in Rep IV uses triadic elicitation to ask an expert how two elements are alike and differ from a third. Then, it asks the expert to create a construct and some poles for that construct. Next, the tool asks the expert to rate all elements according to the poles.

After four constructs have been elicited in this way, Rep IV tests the constructs and elements for similarity. If any two are more than 80% similar, it asks the expert to lower the similarity by either entering a new element or a new construct. If Rep IV gets a new construct, it asks the expert to rate all elements by that construct. If the Rep IV gets a new element, it asks the expert to rate it according to all the existing constructs.

Rep IV can analyze a grid and produce the charts like figures 2.4 and 2.5. Rep IV also can output the raw data used to compute the grid, including the element and construct percent matches. During the elicitation process, the expert can use these charts to decide how to further refine the grid.

Rep IV also facilitates allowing another user to fill in ratings that another expert has generated. This allows comparisons between different experts' ratings.

Overall, Rep IV is easy to use. It is user friendly and robust. Its elicitation feature makes it easy to collect grids from experts. It also performs all the necessary analysis for repertory grids.

2.3.3 Summary

Repertory grids are a technique for eliciting knowledge about similarities from experts. Analyzing the grid produces a numerical percent-similar result that this thesis can use to compare human and computer results. This thesis can include an analysis of the constructs in a grid in order to determine what reasons the respondents had for choosing their constructs and ratings. The technique also has software that can help in

the eliciting and analysis of the grid. Together, this makes repertory grids a technique that is both useful and easy to use in this thesis.

2.4 Creativity

Creativity is a concept that is hard to define and evaluate. Still, there has been much research about how to quantify creativity and how to build systems that exhibit creative reasoning. One kind of reasoning that, if applied correctly, can produce creative reasoning is the kind of analogical reasoning that section 2.2 describes. However, in order to make an analogical reasoning system produce creative analogies, a more precise knowledge of creativity is necessary. In particular, a designer of a creative system must know how to judge if a system produces a creative product and what methods a system can use to produce them. This understanding is critical in this thesis because a one of its goals is to judge if the analogical reasoning system used can produce any creative results.

The following sections describe three important aspects of creativity that can be used to design a reasoning system, such as an analogy making system, that can produce creative products. First, section 2.4.1 describes a set of criteria that can be used to assess creative products. This is useful for judging if a reasoning system has produced anything creative. Second, section 2.4.2 defines creativity in respect to conceptual spaces, which gives another perspective about how to evaluate creative products. It also compares the analogical reasoning approach to other ways of producing creative products. Finally, section 2.4.3 describes how creativity is applied to design tasks specifically for analogical reasoning. This gives more detail about what an analogical reasoning system must do in order to produce creative designs and how useful analogical reasoning can be for design tasks.

2.4.1 Assessing Creativity

Assessing creative products is a difficult skill that requires highly trained judges. Even among highly trained judges there can be disagreement. One kind of disagreement could come from different interpretations of the factors that the judges are using. In one real example, during the evaluation of art, there was a criteria called “merit” [Besemer and Treffinger, 1982]. The merit ratings for two teams of judges had

significant negative correlations.

Besemer and Treffinger attempt to make the criteria for judging creative products less ambiguous [1982]. By doing so, it should allow judges to make more accurate assessments of creative products and even allow people to train themselves to become more creative. They break down the criteria into three groups: novelty, resolution, and elaboration and synthesis with 14 sub categories divided between them. In the rest of this section, the sub categories are written in italics.

Novelty is a measure of the newness of a creative product. The product could have new concepts, new techniques or other new aspects to it. *Originality* refers to the "statistical infrequency" or unusualness of the product. Something that is original, is something that is judged to be infrequent among a certain population. A *germinal* product is considered novel if it has a greater influence on later products. This means that the product allows for later creative products. Finally, a *transformational* product is novel because it presents a new way to look at the world. After understanding the transformational product one might think, "the world will never be the same again."

Resolution is the correctness or rightness of the product to the solution. The resolution of the product can be *logical*. This means that it is consistent with the facts and is a valid solution. This kind of product must still be new and hard to invent. *Adequate* refers to how much of the problem is addressed by the product. If the problem is particularly difficult, important and experts think there's no way to solve it, then a solution that is only adequate is more likely to seem creative. An *appropriate* product is one that solves the problem in a sensible way. The *appropriate* sub category is a basic one, but if a product cannot solve the problem, then it cannot be considered creative. A product can also be creative if it is *useful* and thus has clear applications. It can be *valuable*, if judges evaluate it to be worth some value. The value is a measure that can be defined in different terms. It might be particularly important for judges to agree upon to how to assess value in order to increase consistency among evaluations.

Finally, elaboration and synthesis refers to the style and aesthetic value of the product. An *expressive* product describes how well the product is presented in a understandable manor and how easy it is to use. A *complex* product is seen as creative

if it takes a complex problem and makes it simple. A product is not creative if it is simply complex for no reason. A *well-crafted* product describes how much effort was put into the solution. An *attractive* product is a product that attracts the attention of an observer, not through beauty, but through surprise, humor, or enjoyment. An *organic* product describes a product that is comprehensive, complete, and coherent. Finally, an *elegant* product is an understated or economical solution.

Besemer and Treffinger make several observations about evaluating creative products [1982]. First, a product may be considered creative, even if it does not have a high rating in all 14 sub categories. For example, a product might be highly *original*, but not *elegant*, and still be considered creative. Second, creativity has to be measured with respect to a particular group of products. Third, the more criteria used, the better agreement there should be within a group of judges. Highly trained judges could help to foster agreement, but there is still some ambiguity. Therefore, defining specific creative criteria is meant to reduce the ambiguity.

2.4.2 Conceptual Domains and Creativity

The previous section mentioned some ways of evaluating creative products and started to define some different types of creativity. It also mentioned that a creative product must be evaluated by a group or within a certain domain in order to determine how creative it is. Boden further defines some ideas about creativity and describes examples of systems that can perform creative reasoning [2003].

Boden proposed that a creative idea can classified as either *Psychological creativity* (P creativity) or *Historical-sociological creativity* (H creativity). A P-creative idea is new with respect to an individual. An H-creative idea is new with respect to any idea ever created. All H-creative ideas are P-creative ideas because if the idea is creative with respect to all individuals, then it was creative for the one individual who thought up the idea. Thus, the ideas of P-creativity and H-creativity are used to define the scope of the creative idea in terms of what kind of group the creative product is creative relative to.

As in the previous section, creative ideas are shown to be relative to other ideas within a certain domain. Boden calls this domain a *conceptual space*. A conceptual

space is a culturally familiar domain like music, physics, or story telling. It is defined by a set of enabling constraints which make the structures within it possible. For example, the conceptual space of chemistry would have particular rules for how molecules react. When the constraints are changed, the space is transformed and concepts that were impossible become possible.

There are two ways this space can be explored. One way is through *combinational creativity* which involves the combining or associating between a set of known ideas. It involves techniques such as association and analogy. These methods are used to make comparisons between concepts that already exist within the conceptual space. An example of a creative association would be noticing similarities between things that are different such as "the sun is like a lamp" or recognizing something despite noise, such as recognizing an amateur's drawing of a famous painting.

Analogy is different from association because it performs more deep reasoning about any two particular ideas. It is a more sustained comparison between the internal structure of the two ideas.

The other kind of creativity is called *Exploratory-Transformational creativity* or "ET creativity." It is broken up into two types E and T. E-creativity involves tweaking the conceptual space to achieve creativity. In an E-creative system called AARON, paintings are drawn using a genetic algorithm which tweaks the drawing parameters. The result is a set of similar looking, but novel pictures. When AARON draws acrobats they always have 2 arms, but they might be different in terms of how big they are and their orientation. The systems would never draw acrobats with one arm because its conceptual space does not allow it.

If AARON were T-creative, it would be able to change much more than just the number of arms in the drawings. It would be able to change the overall style of the drawings making something different but related. Thus, T-creative systems differ from combinational and E-creative systems because they can change the conceptual space beyond finding unusual ways of thinking within a conceptual space and tweaking a conceptual space's superficial dimensions.

T-creative systems also have the additional challenge of being able to alter their way of evaluating their creative products. Because they must change what is legal to

express in their conceptual domain, they must also be able to change their evaluation criteria.

ET-creative systems differ from analogy systems in the way they reason about the conceptual space. Both consider the structure of the concepts in the conceptual space, but analogy is focused on the individual concepts that exist, while ET-creativity is concerned with the styles of thinking that exist in the domain.

2.4.3 Creativity in Design

A problem space is defined by the reasoning goal and the operators that enable state space search [Goel, 1997]. If the design variables and their ranges in the problem space remain fixed throughout the design process, it is called *routine design*. If the ranges can change, it is called *innovative design*. If the design variables can change too, it is called *creative design*. The type of design can also change depending on what the designer knows as well. Thus, if the designer knows the structure of the design space and the procedures for searching it, it is routine design, if the designer knows only the structure then it is innovative design, and if the designer knows neither it is creative design.

Creativity in design occurs to different degrees depending on the state of solution and how much knowledge has been transferred from other sources. This transition from creative to routine design happens because, at first, designers may be radically changing the solution space and may even add new knowledge to the design space using techniques such as analogical transfer. Once these are in place, the designer may proceed with innovative or routine design, refusing to change the determined parameters.

The analogy process consists of first taking a given problem P_{new} and a possible solution S_{new} for P_{new}. Then the analogy process applies analogical reasoning to be reminded of a familiar problem P_{old} with a solution S_{old}. Finally, the analogy process transfers selected elements from S_{old} to S_{new}. Goel further explores the issues involved in creative design by asking the questions why, what, how, and when [Goel, 1997].

Analogies can be useful for generating a new solution to a design problem by proposing a new design or by modifying an initial design. Analogies could also help

in other tasks, such as elaboration or decomposition of the problem. Thus, there are many reasons why analogies can be useful for designing.

Answering the what question means describing what kind of knowledge gets transferred by an analogy. The type of knowledge depends on the reasoning task. The knowledge could be design elements, components, and relations between components for tasks like design proposition. For a task like reinterpreting a problem, a different kind of knowledge may be transferred. The transfer of strategic knowledge, such as a method for problem decomposition, is also possible.

Answering the how question means providing methods for reminding and transfer. One method is case-based reasoning. Case-based reasoning is useful when P_{old} is very similar to P_{new}, all of the S_{old} can be transferred to S_{new}, and part of it can be modified to fit the P_{new}. However, case-based reasoning may not work for creative design. If P_{old} and P_{new} are so similar, then S_{old} is probably not going to suggest changes in the variables of P_{new}. Thus, case-based reasoning is probably not going to generate creative solutions because creative solutions must add new design variables to the problem space.

To create such design variables analogical reasoners must use generic abstractions to suggest new variables for the problem space. Generic abstractions express the structure of the relationships between objects as well as the features of objects. In design there could be abstractions for things such as geometric structures or even design goals and methods.

Answering the when question requires describing the strategic control of processing, which can occur during different parts of the design process. For example, generic design abstractions can be learned by an analogical reasoner at different stages of the analogy process. They can be learned by using the existing design library before the designer has made an input, or it could be done during the retrieval stage, once the designer was reminded of a design.

Goel describes several systems that perform creative analogies, two of which are DSSUA and IDEAL.

DSSUA, which was described in section 2.2.4, is creative because it can introduce new variables into the initial solution [Qian and Gero, 1992]. For example, in a door

design problem DSSUA was able to add a variable of sliding motion into the design of a sliding door based on a comparison to a window curtain.

IDEAL is also able to add variables to designs through generic design patterns [Bhatta and Goel, 1996]. The generic design patterns allow for cross domain transfer, which introduces the new variables to the design space.

Goel proposes a research agenda which includes determining what kind of knowledge representation is appropriate for enabling the more efficient processing of generic design abstractions.

2.4.4 Summary

Creativity has various groups of categories including novelty, resolution, and elaboration and synthesis. Evaluating creativity can be improved by giving judges specific criteria. This means that this thesis needs to be careful about which particular groups of creativity it is evaluating and must specify to any judges the criteria they should use.

Boden's work, described in sections 2.4.2, and Goel's work, described in 2.4.3, both describe the concept of a space, called a conceptual space or a problem space, where the range of possible products is specified. The more the reasoner is able to change this space, the more creative the products will be.

From Boden's perspective, the analogical reasoner used in this thesis is not be involved in any dramatic changes in the conceptual space because all the possible products are defined by combining existing products. Thus, the analogical reasoner is it is capable of combinational creativity.

From Goel's perspective the analogical reasoning system is capable of creative design because it can add new variables to the problem space. Since this thesis uses test examples from different domains, the system should be able to introduce design variables from the source domain that did not exist in the target domain.

Also, Goel describes how creativity can be useful for many different design tasks and therefore, if this research can have an impact on making design tools better at being creative, it would be widely applicable.

3. Knowledge Representation

The experiments described in this thesis use SME as the analogical matching algorithm. Since SME uses a symbolic approach, the knowledge representation (KR) must be defined in terms of symbols. In addition to this, the experiments requires the KR to be used to describe some physical objects. Section 2.2.2 contains details about the SME algorithm.

This section describes the goals of the KR and the primitives it uses. It also describes the "functional basis", which provides the set of domain specific terms for representing physical devices. Lastly, this section describes how this KR compares to other function representations described in the literature.

3.1 Requirements and Design Decisions

There are several requirements for the knowledge representation:

1. It must represent DC and EC functions.
2. It must represent devices at different levels of detail.
3. It must allow the DC and EC parts to be combined to form a "BOTH" representation (see example in section 3.3).

The KR must be descriptive enough to describe functions and must allow for different experiments. These experiments (sections 5 and 6) require the ability to represent devices at different levels of detail, and also to use the DC only, EC only, or BOTH versions of the each device's representation.

We consider a function to be a set of desired behaviors. Rather than including all of the constructs from Chandrasekaran and Josephson's work [2000], such as mode of deployment, the KR represents only behaviors and functions, leaving further exploration of Chandrasekaran and Josephson's concepts to future work.

The KR is somewhat independent from SME concepts, but is still easily translatable. This decouples the KR from the particular intricacies of the matching algorithm implementation used.

3.2 Primitives in the Representation

There are five main primitives in the KR: devices, functions, behaviors, relations, and flows. To completely specify a device using the KR, one must provide a library of relations and flows, a set of behaviors and a set of functions that group the behaviors.

A device has a set of functions that are either DC or EC. Each function consists of a set of behaviors. Since a device may have multiple functions, some of a device's behaviors may be mentioned in more than one function.

Devices are physical objects in the world and their behaviors describe how they interact. Behaviors are instantiations of relations. The relations (e.g., import) provide constructs that are filled in with domain specific elements, such as flows or other devices, in order to specify a behavior. For example "import <flow> <device>" is an example of a relation with two arguments. Instances are import torque gear and import force drum.

Flows are the material, energy or signals involved in a particular behavior. For example, a behavior change force surface describes how the flow "force" interacts with the device "surface".

The environment for a particular device is an outer environment defined by a set of external objects that interact with the device. It is *not* the entire external environment. The representation does not have an explicit representation of the complete environment. Instead it describes the environment using behaviors. For example, the behavior transmit torque minutegear references "minutegear," which is part of the environment. Also, the representation can have behaviors that do not refer to the environment at all. To distinguish objects which are part of the environment from the device we mark objects in the environment by underlining them.

3.3 Using the Knowledge Representation

The primitives described in section 3.2 can be used to satisfy the goals we had for the knowledge representation. This section provides examples of devices represented with high and low detail. This section also provides examples of DC and EC behaviors and functions.

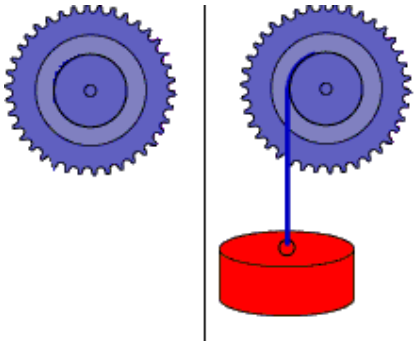


Figure 3.1a (left): A gear. Figure 3.1b (right): A gear and a weight. Other devices that interact with these two, earth and gear2, are not shown.

The KR can be used to represent DC behaviors and functions for the gear pictured in 3.1a. The relation "import <flow> <device>" is used to define the behavior:

import force gear (b1)

The relation "export <flow> <device>" is used to describe the result of behavior b1:

export force gear (b2)

The two behaviors combine to form a single DC function.

b1, b2 (dc1)

To represent EC behaviors and functions, the representation needs to introduce another device to interact with the gear because EC behaviors need to mention something in the gear's environment.

For the situation with a weight and two gears, partially represented in figure 3.1b, two EC behaviors are available for the gear:

transmit force from weight to gear (b3)
transmit force from gear to gear2 (b4)

The environment of the gear consists of weight and gear2. The behaviors b3 and b4 combine to form an EC function:

b3, b4 (ec1)

The weight in the mechanism can also be represented with two behaviors:

transmit force from earth to weight (b5)
transmit force from weight to gear (b6)

The environment of the weight consists of earth and gear. The behaviors b5 and b6 combine to form an EC function for the weight:

b5, b6 (ec2)

When representing with low detail, the representation focuses on a particular device. A device has no internal components and the behaviors for the device either refer to the device itself or to objects in the environment.

For a high detail representation, the KR needs to combine low detail descriptions together. The representation does this by combining the behaviors and functions from several low detail devices. For example, the gear and weight from figure 3.1b could be combined to form a high detail device called "powerprovider". The EC function of this new device would contain four behaviors instead of two and only describe one function. The EC function would be:

b3, b4, b5, b6 (ec3)

This KR can be used to create a BOTH representation by concatenating the EC and DC version of each device representation. Thus, the BOTH representation for the gear consists of the functions dc1 and ec1 as well as the behaviors b1, b2, b3, and b4. Note that as the DC and EC representations use different relations there is no overlap when constructing the BOTH representation.

3.4 Functional Basis

The terms used to describe the function of different devices must be consistent and at the same level of abstraction so that device descriptions are comparable. This will reduce the variation and noise in results. For example, using more abstract terms for one device may cause SME to generate more matches, making strong conclusions harder to make, while inconsistent terms may cause fewer matches, with similar consequences.

This thesis uses a set of domain specific terms called the "functional basis," which was described in section 2.1.5 [Stone and Wood, 1999]. The functional basis provides a set of domain-dependent terms for flows and functions. The representations in this thesis use flows in the same way the functional basis does. The functional basis

represents flows of material, energy or signal that transfer from one device to the next. The basic functions available include import, export, transmit, couple, display, rotate, and change. Our representation uses the basic functions from the functional basis work as a way of describing device behaviors.

3.5 Comparison to Other Research

Overall, our KR is a simpler representation than many others that are described in the literature because it focuses only on behaviors and functions. The KR uses Chandrasekaran and Josephson's ontology as a way to define behaviors and to differentiate DC and EC functions. In their view, however, our EC representation would viewed as mixed since it allows the mention of the device. A pure EC representation would not mention the device at all. For example, our representation permits the EC behavior: transmit force from weight to gear. To make this a pure EC behavior, it would need to change to become: transmit force to gear. In this pure EC representation the gear is part of the environment and the behavior is still a behavior for the weight. Also, the KR does not use all of the parts of Chandrasekaran and Josephson's ontology. For example, it does not contain an explicit representation for modes of deployment. Adding modes of deployment would add extra detail about how the devices are embedded in the environment. However, this was not necessary for our experiments.

Based on the work of Prabhakar and Goel [1996a][1996b], described in section 2.1.2, our EC representation describes the interaction between the device and its outer environment. Our outer environment is defined as the set of objects with which the device interacts. Our KR does not model the external environment.

Rosenman and Gero [1998] and others have described the concept of a purpose and the concept that a device could have several purposes depending on the design situation. This is an important concept in functional reasoning since a function only exists to fulfill some purpose. However, our KR does not explicitly mention purposes. It simply assumes that all functions have some implied purpose. Because there can be more than one implied purpose, our KR allows a device to have more than one function.

Rosenman and Gero also describe a way of describing devices in terms of their structure, behavior, function (SBF). Our KR is only concerned with the BF part of that relationship. The KR assumes that the underlying structure is already there. This is an acceptable assumption because our analogical reasoning does not try to reason about structure or determine how a device works. If it did, then the KR would require a representation of structure in order to determine if behaviors were possible.

Another piece of research this thesis takes advantage of is the ideas suggested by Chandrasekaran [2005], which are described in section 2.1.6. Our KR combined the functional reasoning research, which has defined various KRs, with the functional modeling approach described in the functional basis. This shows Chandrasekaran was correct that the two research streams can be used together and that they can be complementary to each other.

4. Experimental Test Setup

Our experimental system required three parts: a set of test examples, an analogical reasoning algorithm, and a "test harness". The test examples were represented using the KR. The test harness executed the experiment by preparing the test examples, executing analogical reasoning, and analyzing the results. This section describes these different pieces.

4.1 Test Examples

The requirements for the test examples to be used are that they: must have varied levels of detail; must include both DC and EC representations; should be similar enough to allow analogical matches; should allow for novel matches; must be a large enough sample so that general conclusions can be reached; and must be capable of being understood by humans.

The test examples used in this thesis are a set of clocks, which are ideal for satisfying these requirements. Clocks can be decomposed into components and subcomponents. By combining different subcomponents together, the level of detail can be adjusted. Because different types of clocks share component types, there are obvious analogical matches that SME can make, providing good contrast for results that people may consider novel. The test examples represent 21 individual subcomponents, which can be grouped into 8 larger components.

4.1.1 Clock Test Examples

We use two kinds of clocks: a pendulum clock, such as a grandfather clock, and a digital clock, such as a bedroom alarm clock. Each clock has a different way to achieve the functions of setting and displaying the time.

Each clock works differently, but they share common component types and common functions. These components are the powerprovider, which provides some kind of energy into the clock, the timebase, which converts the energy into a periodic signal, a gear, which converts the signal into a once-per-second or once-per-minute signal, and a face which displays the time.

We used articles by Brain [2005a; 2005b] as sources of information about clocks. When using a clock a human needs to observe the time and be able to set the time. Figure 4.1 shows a conceptual diagram of these components and how they interact. Arrows indicate the direction of flow in the clock. For example, the powerprovider transfers energy to the timebase. The human interacts with the clock by resetting it or by receiving a visual signal.

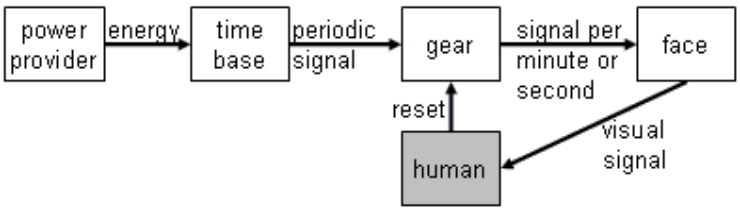


Figure 4.1: Generic model of a clock: components and how they interact with each other and with a human.

Figure 4.2 shows a schematic for a pendulum clock. The schematic labels all the pendulum clock's components. Figure 4.3 shows how these subcomponents get grouped into components. For example, the secondhand and minutehand are subcomponents of face. Figure 4.4 shows the flow diagram for the pendulum clock which indicates how the clock works.

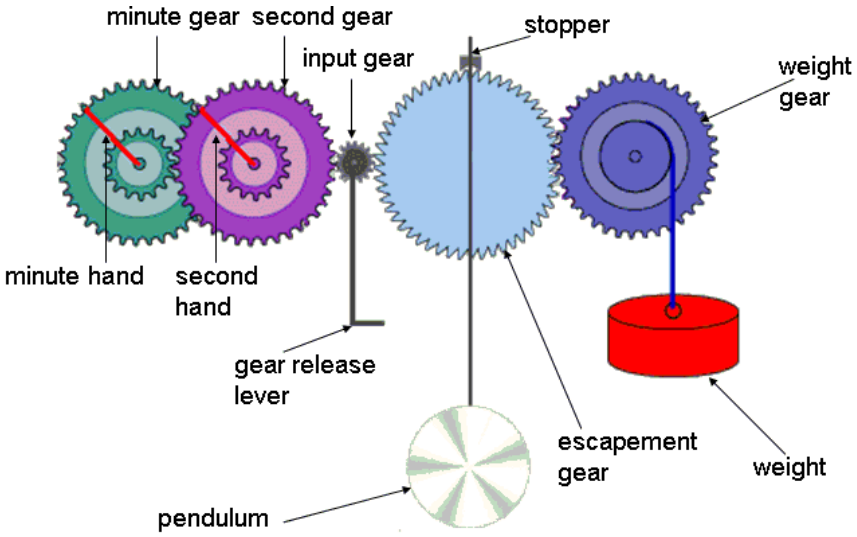


Figure 4.2: Schematic for an idealized pendulum clock showing all its components. Diagram based on [Brain 2005b].

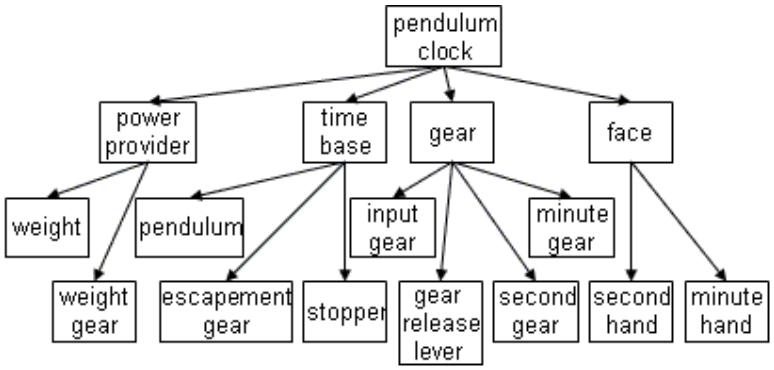


Figure 4.3: Hierarchy for the pendulum clock. Boxes show the devices; arrows represent a component-subcomponent grouping.

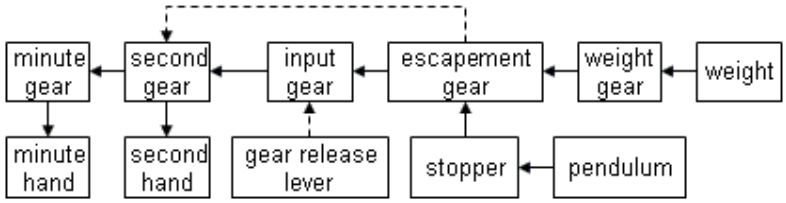


Figure 4.4: Flow diagram for pendulum clock. Boxes represent subcomponents; solid arrows represent flow; the dotted line represents flow when the gear release lever is pressed.

The other clock example is a digital clock. Unlike the pendulum clock, which works primarily with gears, the digital clock uses many divide-by-x counters. Figures 4.5, and 4.6 show the hierarchy, and flow diagrams for the digital clock. A schematic for the digital clock is not provided because it would look very similar to the flow diagram.

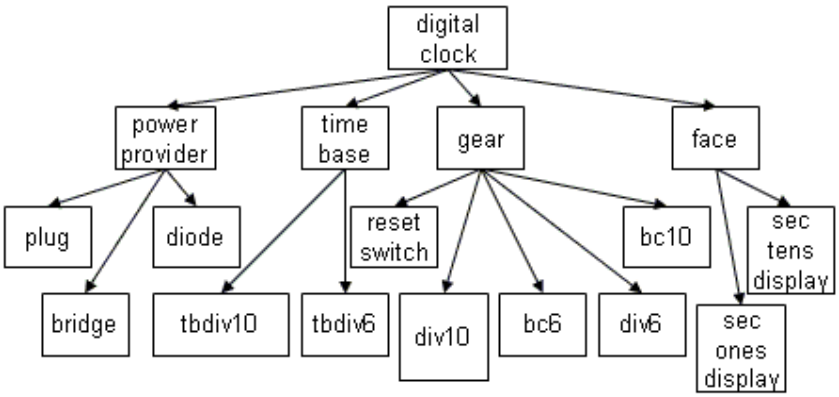


Figure 4.5: Hierarchy for the digital clock. Boxes show the devices; arrows represent a component-subcomponent grouping; bc stands for binary converter; divX stands for divide-by-X counter; tbdivX means divide-by-X counter for the timebase

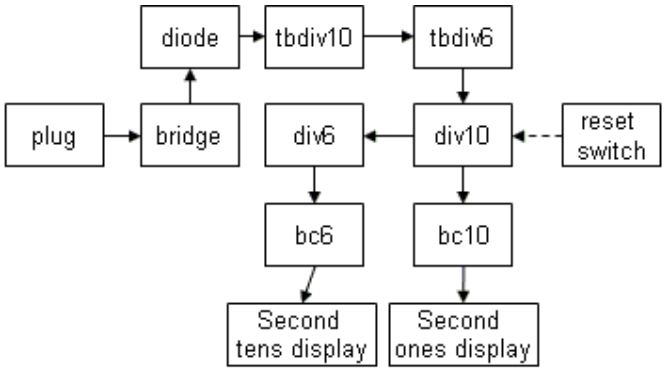


Figure 4.6: Flow diagram for the digital clock. Boxes represent subcomponents; solid arrows represent flow; the dotted line represents flow when the reset switch is pressed. The abbreviations are the same as in figure 4.5.

The hierarchy for the digital clock includes subcomponents such as a divide-by-10 counter, which is part of the digital gear, and a plug, which is part of the digital power provider.

Thus, the test examples are made up of two different clocks that can be represented at two levels of detail. The low detail representations are the subcomponents of the clocks such as secondgear or plug. The high detail representations include clock components such as pendulum timebase and digital powerprovider.

The test examples also have obvious analogical matches both within the same kind of clock and between different kinds of clocks. For example, secondgear could have

an analogical match with minutegear. Secondgear may also be similar to the other gears in the pendulum clock. One reason these analogical matches might happen is because all the gears work the same way, they rotate when torque is applied to them.

Matches between different kinds of clock are also possible and they may appear more novel. They may be more novel because the clock components from different clocks do not work with the same flows or behaviors and they have different surface features. One of the only features they share is their function. For example, both plug and weight both perform the same function of providing energy within their respective the powerprovider components. Thus, there can be an analogy between them, even though one of them works with electricity and the other with gravity.

4.1.2 Applying Functional Basis

This section describes how we applied the terms in the functional basis to model the pendulum and digital clocks. Although the functional basis is intended to be unambiguous, there are still some conventions that we had to follow to apply it consistently to our test domain. Some terms are similar and to apply them consistently we had to be clear about how they would be used. We also had conventions for describing equivalent DC and EC behaviors using separate functional basis terms. Finally, we chose terms that distinguish between changing, transmitting, importing, and exporting a flow.

Tables 4.1, 4.2, and 4.3 describe the flows and behaviors that the test examples use. Each functional basis flow or function in the tables has a description of how we used it to represent a flow or behavior in our test examples. Both the DC and EC test example types use the same kinds of flows, but they have separate behavior vocabularies.

Functional basis flow	Usage	Example
torque	when one device causes another to spin	transmit torque weightgear escapementgear
force	physical energy like gravity	transmit force weight weightgear
signal	when a part of a circuit is sending information over the circuit	transmit signal div10 bc10
electromotive force (eforce)	when part of the circuit is sending just electricity	transmit eforce bridge diode
visual-signal	to represent the flow that was moving from the device to the human's eyes	transmit visual-signal secOnesDisplay human

Table 4.1: functional basis flows used in the test examples

Functional basis function	Usage	Example
rotate	when a device is moving on its own	rotate minutehand
import/export	when a flow comes in or out of a device	import torque minutehand export eforce plug
change	when a flow is already moving between two objects, and then it changes	change signal div10
stop	when a flow stops happening	stop force gear-release-lever
display	showing a visual effect	display visual-signal secOnesDisplay

Table 4.2: functional basis functions used to describe DC behaviors

Functional basis function	Usage	Example
transmit	when a flow is moving from one place to another	transmit signal div6 bc6
change-between	Similar to change, only this one describes two objects. Note: This behavior refers to the "change" function in the functional basis	change-between eforce diode tbdiv10
couple	to describe actively making a physical connection between objects	couple plug wall

Table 4.3: functional basis functions used to describe EC behaviors

Some of the functional basis terms we use appear to be very similar, however the test examples use them in different circumstances. For example, both torque and rotate describe making devices spin, however the device representations use torque to describe when one device causes another to spin and use rotate to describe when a device is moving on its own In describing the digital clocks, device representations use signal when a part of a digital clock is sending information over the circuit and use electromotive force when part of the circuit is sending just electricity.

Since the DC and EC representations have distinct behaviors, there are parallel ways to describe the same kind of behavior. In a DC representation, the devices use import and export to describe flow coming in and out of a device, but in EC representations the device representations use transmit when a flow moves from one device to another. Change and change-between are similar ways of describing the same behavior. However, the change-between, which is used in EC representations, has an additional argument in order to describe an object from the environment.

Finally, changing a flow is different from transmitting, importing, or exporting it. This is an important distinction because it allows the representations to differentiate between two modes of operation. For example, a circuit might normally transmit a signal from a counter chip to a display chip, but when a certain button is pressed on

the circuit board, the representation of the counter chip represents it as a change in the signal between the counter chip and the display chip. Having these extra terms allows the representations to be more complex and precise.

4.2 Applying SME

This section describes how we used SME in this thesis. This involved recognizing which features of SME were important to consider, and converting the KR into a format suitable for the particular SME implementation this thesis uses.

4.2.1 Relevant Properties of SME

Since this thesis is comparing two kinds of KR, there are some properties of SME that are relevant for determining reasons why one KR produces different results than another. These properties are:

- More information in a particular representation should allow for matches of higher weight. This is because longer representations can produce more match hypotheses and thus have higher weighted gmaps.
- Making longer representations may not produce a greater number of gmaps because gmaps can be combined together during the creation of maximally consistent gmaps or because additional information may cause interference when it conflicts with existing information.

Our experiments use these properties to explain why results using the DC and EC representations differ.

4.2.2 Converting Knowledge Representation into SME Input

The experiment needs a way to convert the devices from the KR, which is described in section 3, into the SME format before executing. This input format is specific to the SME implementation this thesis uses. The conversion requires two considerations. First, the input to SME should represent the KR accurately. This means it should be able to represent flows, behaviors, and functions. Second, the input to SME should be set up so that the output is easy to read, otherwise it will take a long time to interpret the results. To fix this, each line in the SME input has only one relation and a unique name.

Table 4.4 shows the mapping between SME concepts and KR concepts. Section 2.2.2 describes the SME concepts in more detail. Converting the KR into SME does not require all the SME concepts. Since the KR only has behaviors and functions relations and entities are all that it needs. SME functions, which are not the same as the functions in the KR, and attributes would be required if the KR had structural relations.

SME concept	**Knowledge representation concept**
dgroup	all the behaviors and functions associated with a single clock component or subcomponent.
entity	a clock component or subcomponent mentioned in a particular dgroup. Also any flow.
relation	instantiations of behaviors and DC and EC functions
function	none
attribute	none
predicate	behavior and function specifications

Table 4.4: mappings between SME and Knowledge representation concepts

The SME implementation requires us to specify a language file that describes all the relations and some of the entity types that are used in the test examples. Relations are specified by "defPredicate" statements and entities are specified by "defEntity" statements. Thus, the language file contains one predicate for each behavior type, and one entity for each flow. The files for each clock component or subcomponent define additional "defEntity" statements. Appendix A.2 shows some examples of complete clock subcomponents converted into SME input.

An example of a behavior definition is shown below. This definition means that there is a relation called "transmit" which takes three named arguments. The first is an entity type called what and the other two are other relations called from and to.

```
(sme:defPredicate transmit
      ((what entity) (from relation) (to relation)) relation)
```

An instantiation of transmit looks like this:

```
((transmit eforce plug bridge) :name *transmit_eforce_plug_bridge)
```

The name part of the instantiation makes the output more readable. The asterisk is not a special character, it is just a way to clearly distinguish names from the other parts of the input. The function predicates are shown below:

```
(sme:defPredicate DC ((predicate)) relation)
```

```
(sme:defPredicate EC ((predicate)) relation)
```

An instantiation of an EC function is:

```
((EC *behavior_set_behavior_couple_plug_wall :name
      *function_behavior_couple_plug_wal)))
```

The EC function has a behavior-set predicate, which is the name of a set of behaviors, and a name, which is unique. Finally, here are some entity definitions which define the flow force and the entity plug:

```
(sme:defEntity force)
(sme:defEntity plug)
```

In addition to statements which map SME inputs directly to KR concepts, some of the predicates were explicitly intended to make the output more readable by putting only one relation on a line. For example, to mark something as a behavior we use this predicate:

```
(sme:defPredicate behavior ((predicate)) relation)
```

Using this makes it so two lines are required to specify a behavior, one to specify the details of the behavior in a relation and the other to mark the relation as a behavior. The benefit of this is that each line of the SME representation has a unique name in the output that marks it as a behavior. Below is an example of how the representation specifies a behavior.

```
((transmit force weight gear) :name  *transmit_force_weight_gear)
((behavior *transmit_force_weight_gear) :name *behavior_transmit_force_weight_gear).
```

To mark a set of behaviors we use this predicate inside function statements:

```
(sme:defPredicate behavior-set (predicate) relation :n-ary? t :commutative? t)
```

Again the reason for introducing behavior-set was so that it could have a unique name in the SME output.

The parameters “n-ary” and “commutative” define important properties for the behaviors specified in a function. They mean that there can be any number of behaviors and that the behaviors can be specified in any order.

We chose not to include attributes of the entities because that would allow SME to make matches based on structure and we wanted SME to make matches based on the behaviors and functions.

4.3 Test Harness

This thesis requires an experimental test harness that allows us to encode test examples, run computational experiments, and perform analysis on both the test examples and the computational test results. The test harness also needs to process the results from the human experiment.

The test harness has several components: test examples, an experiment runner, an SME implementation, a results processor, a repertory grid elicitation program, a grid analyzer, and two analysis tools. There are a variety of technologies that this thesis uses to implement these components. Figures 4.7 and 4.8 show the flow through the test harness as well as the types of technology that were used to implement and represent the various components. The figures also note the format of any intermediate files.

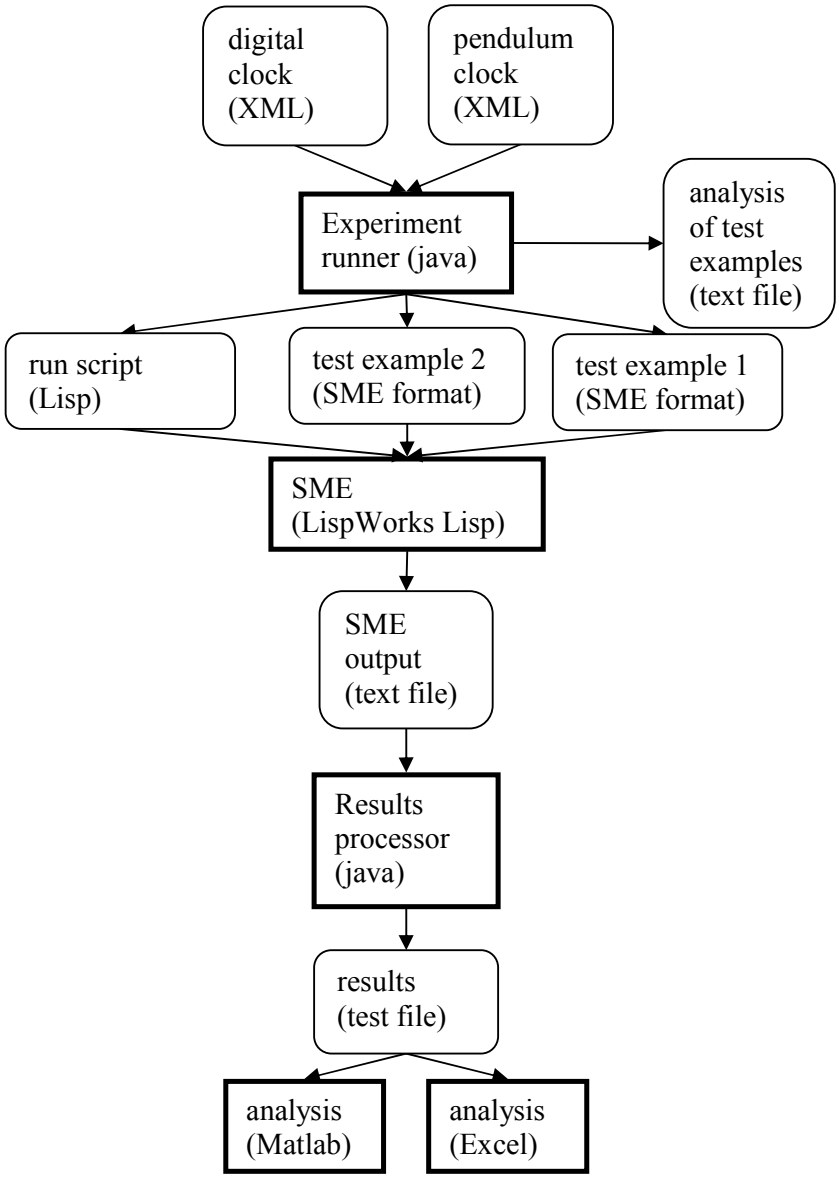


Figure 4.7: Data flow diagram for the experimental system. Boxes with thick edges show the parts that do processing. The format or technology used is in parenthesis. The boxes with rounded edges are pieces of data.

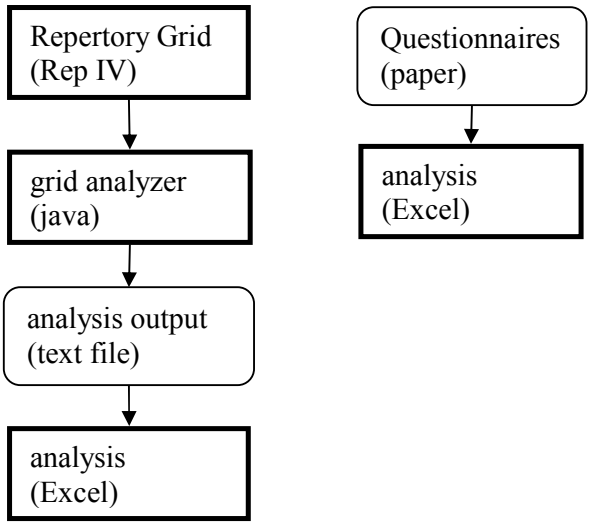


Figure 4.8: Data flow through the test harnesses for human experiment. The format or technology used is in parenthesis. The boxes with rounded edges are pieces of data.

The test harness begins with the test examples, which are expressed as XML files. There is one test example for each type of clock in the experiment. Appendix A.1 shows these input files. The experiment runner reads the files and then performs three operations using the information in them. First, it analyzes them and outputs statistics such as "representation length." Second, it converts the test example representations to a format suitable for SME. Third, it outputs Lisp script that describes a set of device comparisons that SME will run.

The next step is to run SME. This involves loading the Lisp interpreter and running the scripts. The SME implementation is in Lisp because we wanted to take advantage of an existing implementation of SME [Falkenhainer, 2005]. We use LispWorks [www.lispworks.com] as the Lisp interpreter because it has a freeware version and it works with the SME implementation. The result of running SME is a set of text output files. The results processor, reads these files and converts them to convenient formats for Excel and Matlab. Appendix A shows the various input and output formats for some of these steps.

The experiment requires another simpler test harness to analyze data from the human experiment. Figure 4.8 shows the information flow through the test harness.

We typed the output from the repertory grid software directly into the java code for the grid analysis. Thus, the figure does not show an intermediate format for the repertory grid data. Next, grid analysis outputs an analysis of the grid data. Excel reads this data and summarizes it. For the questionnaire, we simply tallied the various results using Excel. Appendices B, C, D, and E show the kind of data that this test harness processed.

In summary, this test harness can be used to orchestrate the computational and human experiments and then analyze the results. The experimental procedures and results will be explained in the next two sections.

5. Computational Experiment and Results

5.1 Experimental Procedure

SME produces a list of gmaps for each match, each with an associated weight. The goal of the computational experiment is to analyze these lists of gmaps and explain how they are affected by different representation types. Overall, the experiment demonstrates the following effects:

- EC has lower weighted matches than DC
- EC has lower weighted matches than DC
- EC generates more matches than DC
- EC matches have higher variance than DC
- BOTH matches are fewer in number and have lower weights than DC or EC alone

The experiment and analysis is able to measure these effects, explain them, and show that they are robust. The experiment measures the gmap weights, the gmap weight variance, and the number of gmaps generated.

The experimental results can be influenced by several factors including the representation length, the representation complexity, and the number of devices mentioned. To make fair comparisons between the datasets the gmap weights and number of gmaps are normalized. Each experiment is run on low and high detail test examples in order to show that any observed effects remain the same even when the level of detail is varied.

5.1.1 Experimental Runs

The experiment uses the factorial experiment design shown in Table 5.1.

	Low detail	**High detail**
EC		
DC		
BOTH		

Table 5.1: Factorial experiment design showing the 6 different device sets.

Overall, there are 6 different device sets that the experiment uses. The rows of table 5.1 show the three versions of device representations, EC, DC, and BOTH. The

columns show that each version is categorized into low detail and high detail. Each combination of a row and column makes up an experiment test set. For example, there is a low detail EC dataset and a high detail DC dataset. There are 21 low detail and 8 high detail devices. The devices are the 21 subcomponents and 8 components of the clocks described in section 4.1.

An experiment test run consists of analyzing pairs of devices from a particular test set. SME compares each device in the test set to the other devices in the test set. The experiment disregards comparisons between the same device. This results in n^2–n comparisons where n is the number of devices in the test set. For example, the low detail test set has 420 matches in it.

There was a problem with using our LISP interpreter to execute SME with one of the devices in the high detail BOTH dataset. Because of this, we omitted that device from the high detail BOTH dataset. To make fair comparisons with this dataset, we made sure that any comparisons we made between the high detail BOTH dataset and other datasets were made with an equal number of devices. This involved removing the device, which could not be processed with SME using our LISP interpreter, from the results only for the purposes of comparing it with the results from the BOTH dataset. For example, to compare the high detail DC dataset with the high detail BOTH dataset, we removed the one device that could not run from the DC and BOTH datasets and then performed the comparison, but when we compared the high detail DC dataset to the high detail EC dataset, we used all devices.

5.1.2 Experimental Factors

This experiment needs to show how the gmap weight, gmap weight variance, and number of gmaps differ for the EC, DC, and BOTH datasets. This is complicated by the fact that several factors can affect these statistics.

The representation length is the sum of the number of functions and behaviors in the source representation. We find that for most of the data, the representation length and the number of gmaps are positively correlated ($p<0.05$). This means that as the representation length increases more gmaps get generated. Our normalization procedure decreases this correlation.

The representation complexity is the sum of the number of behaviors in each function and the number of arguments in each behavior divided by the representation length. For example, the DC version of the gear from section 3.3, with behaviors b1, b2 and function dc1, has a representation complexity of 2. This measure of complexity is similar to the one used in [Balazs, 1999].

In our data, on average, EC representations have the highest amounts of complexity. This is because DC representations only mention the device, and EC representations mention both the device and the environment.

5.1.3 Normalized Gmap Weight and Variance

The experiment needs to compare the magnitude and variance of the weights between the datasets. The factors described in 4.2.1 imply that the gmap weights cannot be compared directly unless some aspects of the representation are taken into account.

Therefore, we use a normalization strategy in order to make a fair comparison between the representations. The normalization formula first computes the value, MAXVAL, which is equal to the highest weighted gmap SME produces when the device is compared to itself. Then the weight of each gmap made with that device is divided by MAXVAL to obtain a new normalized weight.

This strategy should adjust the magnitudes of the gmap weights to account for both the representation length and complexity. It also gives the measurement more meaning. Instead of measuring its overall strength, this normalized weight measures the relative amount of a device's representation that is matched by the target device. Thus, the higher the normalized weight, the more of the target device fits with the source device.

Each time SME generates a match it outputs a list of gmaps, each of which has an associated weight. Since our comparisons are done on a per match basis and not on a per gmap basis we need to aggregate the gmap weights for each match and then use the aggregated result for our analysis. Thus, for each match, we compute the average, standard deviation, and highest of its gmap weights. Then, for all matches we compute additional statistics to create results such as "average of average gmap weights" or "average standard deviation of gmaps."

5.1.4 Normalized Number of Gmaps

The number of gmaps is positively correlated with the representation length. In order to account for this influence and to compare the different datasets, we normalize the data by the representation length. Unlike the gmap weight measure, we could not use the number of gmaps generated when a device is compared to itself because it was not close to an upper bound on the number of gmaps.

The formula for computing the normalized number of gmaps is the number of gmaps divided by the representation length. For example, if a match has a representation length of 5 and generates 10 gmaps, then the normalized number of gmaps would be 2.

5.2 Computational Results

Tables 5.2 to 5.5 show the averages of the computational results for the various measurements in the experiment. Higher values indicate a stronger match. Appendix A.4 shows the raw data used as well as additional summary statistics.

	Low detail	**High detail**
EC	0.5543	0.4705
DC	0.6907	0.5580
BOTH	0.4390	0.3935

Table 5.2: average of average normalized gmap weights per match

	Low detail	**High detail**
EC	0.7629	0.6460
DC	0.6907	0.6086
BOTH	0.7081	0.6233

Table 5.3: average highest normalized gmap weight per match

	Low detail	High detail
EC	0.1796	0.1212
DC	0.0	0.0435
BOTH	0.2512	0.1275

Table 5.4: average standard deviations of normalized gmap weights per match

	Low detail	High detail
EC	0.9421	2.5664
DC	0.2952	1.2389
BOTH	0.4883	1.9624

Table 5.5: normalized number of gmaps per match

5.2.1 DC and EC Comparison

Our hypothesis concerning gmap weights was that the DC weights would be higher than EC weights. This is true for average gmap weight, but not true for highest gmap weight. The difference for low detail result (Table 5.2) is statistically significant ($p<0.05$), while the difference for high detail is not.

This can be explained by the standard deviations in Table 5.4: it shows that the standard deviation for EC is higher than it is for DC. The standard deviations are statistically different ($p<0.05$). Although the EC representation might have a few gmaps with higher weights, it has other lower weighted gmaps that decrease the match's average gmap weight. Thus, in the experiments DC representations produced a few high weighted matches that have similar weights while, the EC representations produced matches that have a wider variety of weights. This resulted in lower average gmap weights and higher highest gmap weights for the EC representation.

Another one of our hypotheses was that EC would produce more matches than DC. The data, shown in Table 5.5, shows that EC produces at least twice as many gmaps as DC. This result is statistically significant ($p<0.05$).

5.2.2 BOTH Dataset

Our final hypothesis is that the matches from the BOTH dataset will have more matches of higher weights than the DC or EC datasets. This makes sense because the more information the representation has, the more it should be able to match.

Our results show the hypothesis is correct for absolute gmap weights, but not for the normalized weights. The normalized weights measure how much of the representation was matched. This result means that a large portion of the BOTH representation is left unused in each gmap.

We observed that gmap weights from the BOTH dataset have a lower highest gmap weight than the ones from the EC dataset and only a slightly higher highest gmap weight than the ones from DC dataset. The EC dataset has statistically different highest gmap weights and the DC dataset does not have statistically different highest gmap weights.

We also observed that the average gmap weight for the BOTH dataset was lower than it was for the DC and EC datasets. This effect is partly caused by the fact that BOTH has a higher standard deviation than DC. However, this does not explain the difference the BOTH dataset has with the EC dataset, because they have about the same standard deviation. A statistical test did not reject the possibility that the standard deviations are similar.

One explanation for this is that when DC and EC information are together the DC information is preventing the matches that would have been generated if only the EC information was present. It could be that with the BOTH representation it is harder to make globally consistent gmaps, as there is so much data with which to be globally consistent. Because the normalization discounts for not having large matches, the match weights are lower.

Another observation about the BOTH dataset is that its highest gmap weight and number of gmap measures are in between DC and EC measures. It seems that adding EC information to the DC information improved the highest gmap weight and number of gmaps by only 39% and 55% of what would have been gained by using the EC information only. Investigating this further, we found that the number of gmap weights from the BOTH dataset is not statistically different from a dataset made by averaging the number of gmap weights from the DC and EC datasets. The average number of gmaps for the averaged DC/EC dataset is 1.8245, which is close to the value of 1.9624 for BOTH.

5.2.3 Robustness to Level of Detail

With a few exceptions, these observations are robust to changes in the detail of the representation. The data shows that the same trends occur in the low detail as in the high detail data. The observations that are different are caused by special properties of the low detail data. One difference is that the DC representation seems to be less effective in low detail devices than in high. The low detail DC representations produced one gmap at most for any matches. The high detail representation, however, did not have this problem. We conclude that the reason for this is that the low detail representation did not have enough information to produce many potential matches.

6. Human Experiment and Results

In the computational experiments we present SME with representations of two devices, and it outputs a list of potential matches between portions of each representation. For example, based on these lower level matches, SME might suggest that a pen is like a hammer.

As our hypothesis concerns the possible benefits of different styles of device representation, the representation is varied throughout the experiments, and the resulting matches are measured and evaluated.

We are interested in performing an experiment with human respondents, which examines these computational results, for two reasons. First, we would like to determine whether or not the matches proposed by SME are "novel": e.g., a pen is like a sponge. We hypothesize that EC device representations are more likely to produce novel matches.

Second, we would like to investigate how the match weights generated by respondents correlate with SME match weights. There are two ways the SME results can correlate.

First, the human and SME results could place the same relative weights on certain device matches. For example, both the human and SME could think that the pen is more like a hammer than it is like a sponge.

Second, the human results can lend support to the DC or EC representation if the reasons the humans are using match with the representation that SME uses, and if the human's and SME's match weights correlate. We might get this result if the human thought the pen was most like a sponge because they both interact with liquid and if SME marked them as most similar because the pen and sponge both interact with a human's hand. Though the reasons are not exactly similar, they both involve EC reasoning: i.e., about how the device interacts with the environment.

To gather this information from human respondents, we use two techniques: repertory grids and a questionnaire [Hart, 1986]. The respondents are a volunteer group of engineers.

The repertory grid technique provides several benefits:

- It is a proven technique that allows respondents to give information about the similarity of different devices in a group.
- The result of collecting the grid information is a "percent similar" measure describing the human's evaluation of device similarity. After normalization, it can be compared to SME output which also reports how similar devices are.
- As part of the grid creation process, respondents give reasons why they differentiated one device from another. This information can be classified as DC or EC, lending support to that approach. It can also be compared directly to the lower level matches in the computer results.
- A good computer tool is available that makes collection of repertory grids relatively easy [Shaw and Gaines, 2005].

6.1 Experimental Procedure

The experimental procedure provided a way for human respondents to describe similarities between different clock components and a way for them to rate the novelty of matches made from those clock components. To complete the procedure, the respondents first read information about clocks, then completed a repertory grid, and finally filled out a questionnaire. This section describes rationale and additional details about the human experiment. For details about the repertory grid technique see section 2.3.

6.1.1 Experimental Setup

Setting up the human experiment required several considerations. The experiment addresses these considerations while trying to make sure that the respondents could perform well and the experiment did not take longer than one hour. First, the experiment required a set of clock components that would allow the computer and the human to discover a wide range of similarities. The set of components also needed to be small enough so the experiment did not overwhelm the respondents with too much information. Second, the respondents needed to understand some background information in order to make judgments about the clock components. Finally, the experiment needed to be sequenced such that the respondents would perform best on each task.

The experiment used six clock components that were identical to the ones used in the computational experiment. These were: digital power provider, digital timebase, digital gear, pendulum power provider, pendulum gear, and pendulum face. There are several reasons why the experiment used clock components and why the experiment used these particular clock components.

Preliminary experiments with simpler examples, such as pens and sponges, indicated that the respondents were using reasons that we could classify as DC or EC. However, many of their reasons were focused on surface features. The clock examples subsequently adopted have similar functions, but very different surface features. Therefore, the respondents tend to focus their attention on the function of the clock components, which is what we want. For example, one respondent created a repertory grid construct called "display converter" which is refers to the function of the clock component.

The experiment used an equal number of components from the pendulum and digital clocks. This meant that the respondents did not have an advantage if they understood one clock better than another.

The set of clock components contained either one pendulum or digital clock type or both types for each generic clock component. Table 6.1 shows which types were included in the set of clock components.

Generic clock component	**Types included (Pendulum and/or digital)**
power provider	pendulum and digital
time base	digital only
gear	pendulum and digital
face	pendulum only

Table 6.1: shows the type of clock components included in the human experiment.

The experiment included both versions of the power provider and gear, but only the digital version of the time base and the only pendulum version of the face clock component. Having both similar and dissimilar versions of components allowed the possibility of the respondents observing a wider variety of similarities and differences. For example, the respondent observations could be "both versions of the gear convert a signal to a periodic signal, but the pendulum one uses mechanical force" or "the face is observed by an external entity but both versions of the gears are involved in the

internal workings of the clock."

Keeping the number of clock components to a minimum kept the respondents from being overwhelmed and also helped keep the experiment time short. If the respondents had to consider too many devices they might not have been able to think as deeply about the devices they were evaluating. Thus, keeping them from being overwhelmed should increase the quality of their evaluations. Since the respondents took time to evaluate each component based on each of the six constructs they created, adding more components increased the overall repertory grid collection time. In an experiment with six clock components, the respondent would have to make 36 ratings, but with eight it would be 48. Our observation was that it took the respondents about 1 minute to come up with a construct and then a least 30 seconds to create an average rating. Thus, reducing the number of components saves about 6 minutes of respondent time.

The second consideration the human experiment required was ensuring the respondents had adequate understanding of how clocks work. To accomplish this, the respondents had to read articles about clocks are part of the experiment. These articles were the same ones that we used to create the representations for the computational experiment, taken from *How Digital Clocks Work* [Brain, 2005a] and *How Pendulum Clocks Work* [Brain, 2005b]. Also, during the experiment the respondents had a schematic for the pendulum and digital clocks that they used throughout the experiment. Appendix B shows these schematics. Giving them all this information should bias their comparisons. This was intentional because we wanted the respondents to use the same information the computer used as much as possible.

Since the respondents were engineers, they had little trouble understanding the examples, given the documentation. Despite this, they were still allowed to ask questions to clarify the schematics or their understanding of the clocks. For example, they asked clarifying questions such as "are the two div10's the same object or are they different?" or "what does tb mean"? They also asked for simple explanations of what the diagram was showing like "I don't understand what the stopper is for." In general, however, the respondents understood the clocks well and only asked one or two questions during the experiment.

The last the experimental consideration was determining the order in which each respondent would complete the tasks. The experimental procedure was to first allow the respondents to read the background material about clocks. Next, the experimenter collected a repertory grid and then had the respondent fill out a questionnaire. Giving them the articles about clocks first gave them the background knowledge they would need in the experiment. Collecting the repertory grid before the questionnaire was important because it allowed the respondents to determine for themselves how the devices relate to each other. Thus, when they filled out the questionnaire, they were able to compare the computer's answers to their own and be better able to judge the novelty of them.

This sequence seemed to work well for allowing the respondents to have the right knowledge to perform the experimental tasks. We found that the allotted 15 minutes was an adequate amount of time for the respondents to read the documentation because most were done reading the documentation within that amount of time.

The respondents were able to understand the workings of the clocks because the respondents seemed to have the right kind of mental models for the clocks. They mentioned words like "divider" without needing prompting from the experimenter.

Our observations of the respondents filling out the questionnaire show that performing the repertory grid before the questionnaire was advantageous. Many of the respondents were able to fill out the questionnaire quickly, in less than 10 minutes. Also, some of the respondents, such as #7, did not need to refer to the schematics while they filled out the questionnaire.

In summary, the experimental procedure had 3 steps:

1. Respondents review material on clocks for 15 minutes.
2. Respondents complete a repertory grid.
3. Respondents complete a questionnaire.

The next two sections will describe steps 2 and 3 in more detail.

6.1.2 Repertory Grid Collection

After looking over the documentation about clocks the respondents' second task was to complete a repertory grid using the Rep IV software. This involved three steps,

introducing the respondent to the software, collecting the first four constructs, and collecting the final two constructs.

Rep IV is not a complicated piece of software to use. The main challenge for a respondent is to know what kinds of questions it will ask and how to select constructs properly. To give the respondent this understanding, the experiment required the respondent to practice completing two constructs of a grid for some practice data about fruits and vegetables. During the practice, the experimenter clarified that the respondent should try to select constructs that applied to all items. This practice helped to get the respondent familiar with the software.

After the respondent completed the practice grid, the respondent began with the real experiment by collecting the first four constructs from the clock components dataset. The Rep IV facilitated this by asking the respondent to evaluate three elements at a time. These sets of three elements are called triads. Each element in Rep IV was a clock component. Every respondent received the same first four triads to generate the constructs. These four triads were dependent on the order in which the clock components were entered into Rep IV.

Each respondent except respondent #1 received the clock components in the same order. Respondent #1 was a special case because he was the first respondent. After observing the results from respondent #1 and respondent #2, each respondent consistently received the elements the order that was used with respondent #2.

The last two constructs were different for each respondent. As described in section 2.3.2, if Rep IV determined that two elements were more than 80% similar, it asked for a clarifying construct. If all elements were less than 80% similar, the experimenter commanded Rep IV to generate an additional triad. The experiment always ended once six constructs were collected.

6.1.3 Questionnaire

The goal of the questionnaire is for the respondents to evaluate the novelty of the computer matches between components of the pendulum and digital clocks. Preparing the questionnaire required several considerations. First, the questionnaire required a rating scale. Second, the questionnaire required a way to present the information so that the respondents would understand how to properly complete it. Lastly, the

questionnaire required some test questions which needed to be based on the gmaps in the computational experiment. Appendix C.1 contains the final questionnaire that was used.

The respondent rated each match on the questionnaire as being low, medium, or high novelty. This rating system has three choices because it allows the analysis to discern when the respondent had a strong opinion about the novelty of the match. If the respondents did not have a strong feeling about the novelty of the match, we assumed that they would select medium novelty. Thus, the analysis should concentrate on the high and low results.

There are two issues with making sure that the respondents understood how to properly make novelty judgments. First, based on Besemer and Treffinger's definitions [1982], which are described in section 2.4.1, novelty is only one type of creativity. Second, Besemer and Treffinger show that giving judges of creativity a set of criteria in which to judge creative products will improve the consistency of the ratings. Thus, the questionnaire needed to be clear about the precise meaning of what novelty meant and how to judge it. Before beginning the questionnaire, the experimenter described the meaning of novelty and gave examples of how to judge the matches they were about the evaluate. These instructions are included in beginning of the questionnaire (Appendix C.1). The main concepts that the respondents needed to understand were that the correctness of the match should not influence their judgment since then they would be evaluating logical correctness instead of novelty, and also that they should rate a match higher if the match was surprising to them or if it was something they would not have thought up easily.

The other issue with making the questionnaire's goals clear was turning the output from the computer into a human readable format that emphasized the match hypotheses that are contained in the match. To accomplish this, the questionnaire separated the concepts in a match into categories and presented them in a table format. Figure 6.1 shows a sample question.

3. digital timebase :: pendulum face

___low ___medium ___high

Devices similar:

tbdiv10	minutehand
tbdiv6	secondhand

Flows similar:

electric signal	torque
electric force	torque

Behaviors similar:

import electric force tbdiv10	import torque minutehand
import signal tbdiv6	import torque secondhand

Figure 6.1 a sample question from the questionnaire.

Figure 6.1 shows the three tables that each question included. The categories were "devices similar", "flows similar", and "behaviors similar." The table format emphasized that the match was saying that the item on the left is similar to the item on the right. For example, the flow "electric signal" is similar to "torque" and the behavior "import signal tbdiv6' is similar to "import torque secondhand."

The selection of questions needed to vary two different dependent variables while minimizing other factors which might add noise to the results. The two dependent variables were whether the match was EC or DC and how much weight the computer assigned to the matches. The questions also had to allow the respondent to make relative comparisons since creativity is a relative judgment. The experiment needed to accomplish this while keeping the number of questions low so that the respondents could complete them in a reasonable amount of time.

Table 6.2 shows the classification for each of the eight questions on the questionnaire.

Question	Weight	Type	Gmap weight
1	High	EC	8.422
2	Low	EC	4.4068
3	High	DC	7.1871
4	Low	DC	11.0076
5	High	DC	13.21
6	Mid	EC	7.406
7	High	DC	14.0506
8	High	EC	8.1836

Table 6.2: question number, SME weight, question type, gmap weight

The questions included an equal number of DC and EC types with a variety of weights. The weights were classified as either high, mid, or low according to their relative gmap weight among the list of gmaps for a particular match. When the chosen match was the highest value in the list of gmaps, it was classified as "high," if the match was the lowest, it was coded as "low" and if there was a match in between high and low, it is coded as "mid". Thus, these measures relate to the relative value of the gmaps, not to the absolute values.

Table 6.2 shows the resulting weight classifications. There are three high DC weight, two low EC weight, one low DC weight, one low EC weight, and one mid EC weight. The distribution of weight types is not ideal, since the type of weight is not evenly balanced between DC and EC representations and the number of high, mid, and low is not the same. However, the selection of gmaps was limited by the selected clock components and by the matches that were actually available. The resulting questions are the best balance of these factors.

The questions are always between components of the pendulum and digital clock types because we thought that those matches would have the greatest potential for being novel since they are cross domain matches.

The selection of which clock component matches to include was influenced by wanting to control the number of dependent variables being varied. One of the dependent variables we wanted to reduce was the effect of having different clock components, since having more clock components would reduce the number of relative comparisons the respondents could make. Thus, all the matches in the questionnaire involved the digital timebase and the questionnaire included multiple

questions comparing the same pair of clock components. For example, questions 1, 2, and 3 all compared the digital timebase to the pendulum face. Questions 1 and 2 were EC, question 3 was DC, question 2 was low value, and questions 1 and 3 were high value. The only difference between question 1 and 2 is the value. The only difference between 1 and 3 is its question type.

Another factor we wanted to minimize was the respondents' unfamiliarity with the clock components mentioned in the questionnaire. Being unfamiliar with the clock components would make it hard for the respondents to judge novelty. The questionnaire minimizes unfamiliarity by containing questions about clock components that the respondents were familiar with from the repertory grid experiment.

A final factor we wanted to minimize was a respondent's bias towards one particular clock component, thus the questions include comparing the digital time base to the pendulum face, pendulum gear, and pendulum power provider.

One potential bias in our questionnaire was the fact that the respondents had much more knowledge than the computer did and therefore, the computer could not possibly come up with matches that respondents thought were novel. In fact, one respondent thought that all of the computer matches were not novel. The computer was at a disadvantage because it only has the knowledge that we encoded in it, which is limited. However, this effect is reduced because the computer's and respondents' knowledge were both based at least somewhat on the same articles. Also, there are not too many different ways to represent the functions and workings of the clocks.

6.2 Results and Analysis

This section contains analysis of the repertory grid and questionnaires collected from ten respondents between November and December 2005. The results include an analysis of the repertory grid constructs and the similarity measures. They also include analysis of the question representation types and question gmap weights. Each section describes the results and how they were analyzed.

6.2.1 Summarizing Constructs

Each respondent contributed six constructs during the repertory grid elicitation. One goal in the human experiment was to try to determine if there was a correlation between the reasons people gave and the computer results. Another goal was to determine if humans were thinking more in DC terms or more in EC terms. To do this, we had to classify all the human constructs as either DC, EC, or neither.

We classified the constructs in a two step process. First, we grouped the constructs into categories without determining their classification. Each of the categories had a central theme. Once the constructs were categorized, we characterized the individual constructs as DC, EC, or neither according to the following criteria:

- The construct is EC if the construct required knowledge of the environment in order to understand or make a judgment based on it.
- The construct is DC if the construct only required knowledge of the clock component to understand or make a judgment based on it.
- The construct is neither if the respondent was making a superficial judgment that did not require much thought. An example of a superficial constructs is: "has two words in it." In this case, the respondent was referring to the way the clock component was represented on the page.
- The construct is neither if the construct was the result from obvious use of our experimental setup. For example, "part of pendulum vs. part of digital" is neither since it was related to an artificial grouping made by the experiment, not by the respondents. It also was part of nearly all the respondents constructs, which confirms that it is an obvious construct.

6.2.2 Analysis of Constructs

The result of this analysis was 16 categories. The respondents had 18 DC constructs, 31 EC constructs, and 11 constructs that were neither. Appendix D.2 contains the categories and listing of the respondents constructs. Figure 6.2 shows the counts of the constructs of each type per respondent for each question representation type.

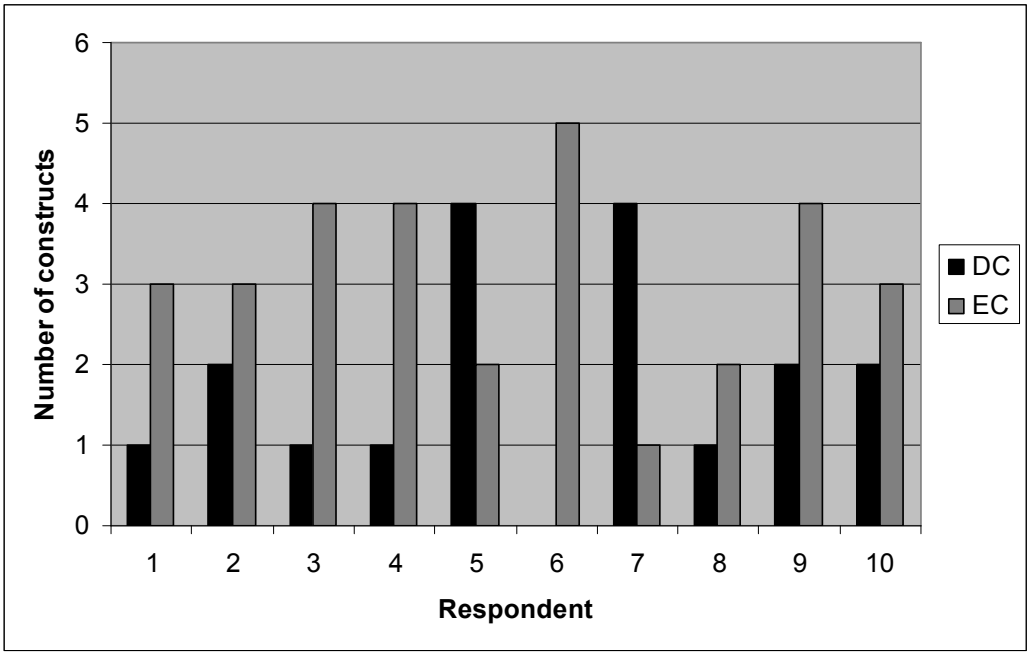


Figure 6.2: classification of respondent constructs per question representation type.

Based on the counts, the respondent who had the most EC reasons was respondent 6. The respondents who had the most DC reasons were respondent 5 and 7.

6.2.3 Percent Similar Analysis from Repertory Grid

We used the "percent similar" measure generated by the repertory grids collected from the respondents, and compared that measure to the normalized highest gmap weight generated by SME. Each repertory grid had 6 devices, making 36 possible evaluations between devices.

For example, figure 6.3 shows the grid from respondent #7.

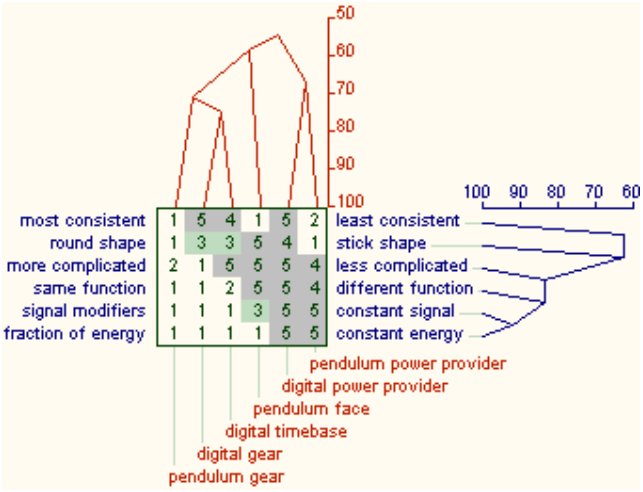


Figure 6.3: grid for respondent #7

Using the formula described in section 2.3.1, the percent similar between the pendulum power provider and digital power provider is 66%. The data for all respondents can be extracted from the grids in Appendix D.1.

Although percent similar can range between 0 and 1, it should not be directly compared to SME data since the repertory grid collection technique asks for clarification when similarity levels are above a certain percent. This makes the percentages artificially low. Therefore, we computed *match rankings* based on the percent similar measures generated by SME and the repertory grid.

The rankings are between the pairs of matches that are possible between the clock components. Table 6.3 shows a sample set of matches and what their rankings would be.

From	**To**	**Rating**	**Rank**
pendulum power provider	pendulum time base	0.5	3
pendulum power provider	digital time base	0.825	1
digital face	pendulum face	0.72	2

Table 6.3: example of how match ratings get turned into ranks by the statistical methods.

We looked for correlations between each of three possible data sets and each of the 10 individual respondents. The data sets are DC, EC, and BOTH. We use the Spearman rank order test to detect correlation between the datasets and the Wilcoxon signed rank for testing that the medians of the differences between the datasets are different. The correlation are shown below in figure 6.4.

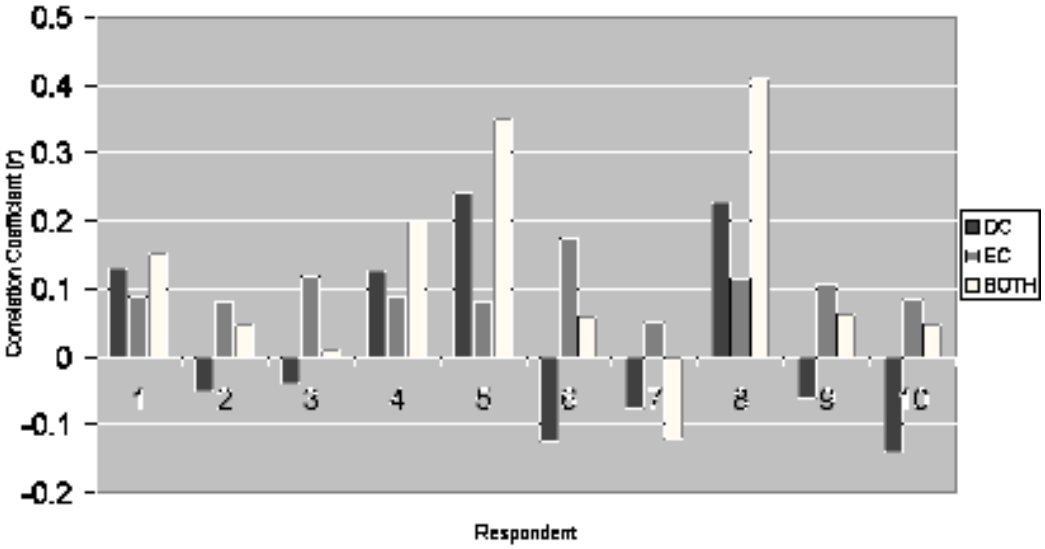


Figure 6.4: correlation coefficients between the respondent and SME datasets.

The tests show no significant correlations between the DC and EC datasets and the respondents' answers ($p>0.23$) and that the DC and EC datasets and the respondents' answers are significantly different ($p<0.1$). Another observation is that, on average, the EC dataset is more positively correlated to the respondent data than DC.

For the BOTH dataset, there are two significant correlations with the answers from respondents 5 and 8 ($p<0.1$). The rest are insignificant ($p>0.32$). The sign rank test shows that the medians of the respondent 5 and 8 datasets are different from the median of the BOTH dataset ($p>0.5$).

Figure 6.5 shows a scatter plot for respondent #8's match rankings. Respondent 8's match ratings had a correlation value of 0.41 which means that the BOTH dataset results can account for 16.72% of respondent #8's responses. Although the correlation can be hard to see, there are some specific cases where results from respondent #8 and the BOTH datasets had a match at the exact same rank. For example, the match at rank 7, which was between the pendulum face and pendulum power, is ranked exactly the same. Sharing the same rank means that both the computer and respondent #8 thought that this match ranked the same among all the other matches.

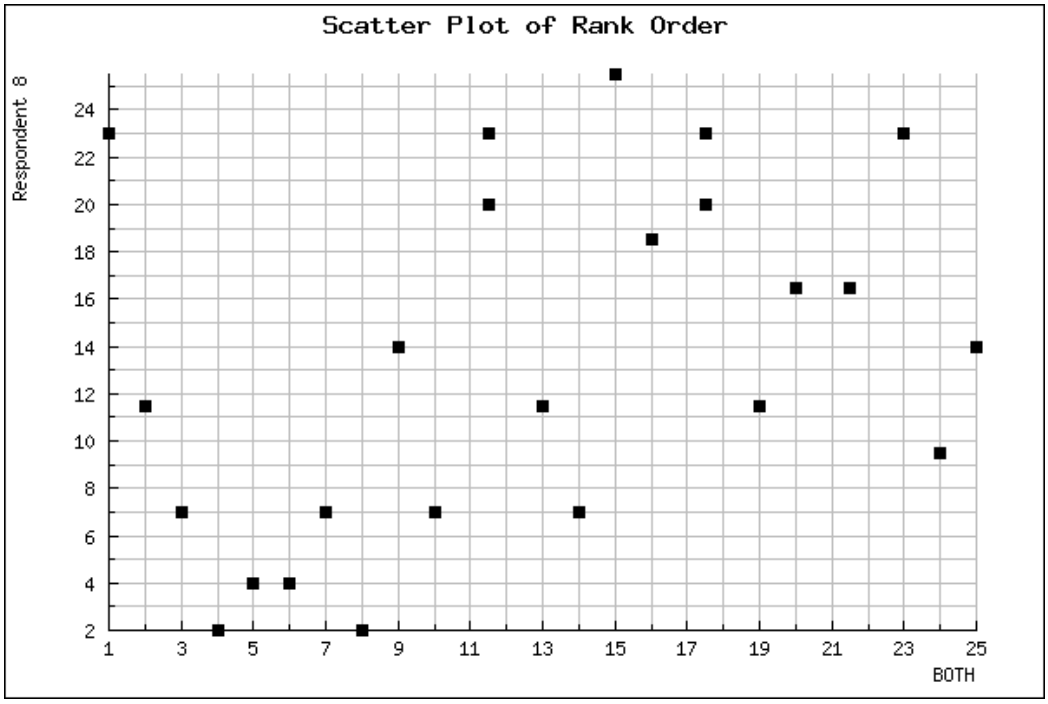


Figure 6.5: rank order correspondence between device matches from respondent #8 and the BOTH dataset. The figure shows, for example, that one of the device matches is ranked 9th in the BOTH dataset, but is ranked 14 in respondent #8's data. Ties in the ranks are possible, which can cause multiple data points on a single column or row.

6.2.4 Respondent Constructs Characterizations Compared to Respondent Correlations

It is interesting to try to determine whether the respondents' constructs given in the repertory grid correlate with the representation type used to generate the SME results. This would enable us to determine if the respondents were thinking more in DC or EC terms.

The correlation coefficients measure how well the respondent's similarity ratings correlate with the results from the DC and EC datasets. Therefore, if the respondent's construct characterizations correspond to the correlation coefficients, we can conclude that the respondent's constructs are more DC or EC and thus the respondent is thinking more in DC or EC terms. We performed three types of analysis to determine this.

The first analysis we did was we compared the respondent who had the highest number of DC or EC constructs with the correlation coefficients. The respondent with

the most EC constructs had the strongest correlation with the EC dataset. Similarly, one of the respondents with the most DC constructs had the strongest correlation with the DC dataset. This kind of correspondence did not always hold, however, the other respondent who was most correlated with the DC dataset had 4 EC constructs and 2 DC constructs.

The second analysis we did was to see if the characterization of the respondents' constructs predicts which dataset they will be more correlated with. The data shows that this occurred in the data from 6 of the 10 respondents.

The first and second analyses only try to determine if the results from respondents are more like DC or EC results. In reality, the respondents' reasons are probably somewhere in between the two extremes. To quantify this better the data requires a third analysis using a statistic that is higher if the respondent's ratings are more EC and lower if the ratings are more DC.

The analysis has a method for computing this statistic for both the correlation coefficients and the respondent constructs. For correlations, the analysis computes this statistic by a subtracting the DC correlation coefficient from the EC correlation coefficient. For the respondent constructs, the analysis subtracts the number of EC constructs from the DC constructs. The resulting data are suitable for a spearman rank order test. Tables 6.4 and 6.5 show the correlation coefficients and the construct characterizations along with the measure of how DC or EC the numbers are. For example, respondent 1 has an statistic of -0.04 which means the similarity ratings are slightly more DC. Respondent 9 has a statistic of 2 which means the constructs are more EC than DC.

respondent	EC	DC	EC – DC
1	0.09	0.13	-0.04
2	0.08	-0.05	0.13
3	0.12	-0.04	0.16
4	0.087983	0.127415	-0.03943
5	0.079149	0.24091	-0.16176
6	0.174911	-0.12282	0.297728
7	0.049245	-0.07517	0.124414
8	0.114019	0.226566	-0.11255
9	0.108787	-0.05996	0.168743
10	0.08424	-0.14062	0.22486

Table 6.4: correlation coefficients of the respondent's similarity ratings for EC and DC. The third column is the EC correlation coefficient minus the DC correlation coefficient. The higher the number the more EC the respondent's similarity ratings are, the lower the number the more DC the ratings are.

respondent	EC	DC	EC – DC
1	3	1	2
2	3	2	1
3	4	1	3
4	4	1	3
5	2	4	-2
6	5	0	5
7	1	4	-3
8	2	1	1
9	4	2	2
10	3	2	1

Table 6.5: construct characterizations of the respondent's constructs for EC and DC. The third column is the number of EC constructs minus the number of DC constructs. The higher the number the more EC the respondent's constructs are, the lower the number the more DC the ratings are.

The results from the spearman rank order computation are that the two statistics have a medium strength, positive correlation ($r = 0.46$) with a significance of ($p<0.16$). Figure 6.6 shows the scatter plot of the ranks of the statistic computed for the correlation coefficients and the respondent constructs.

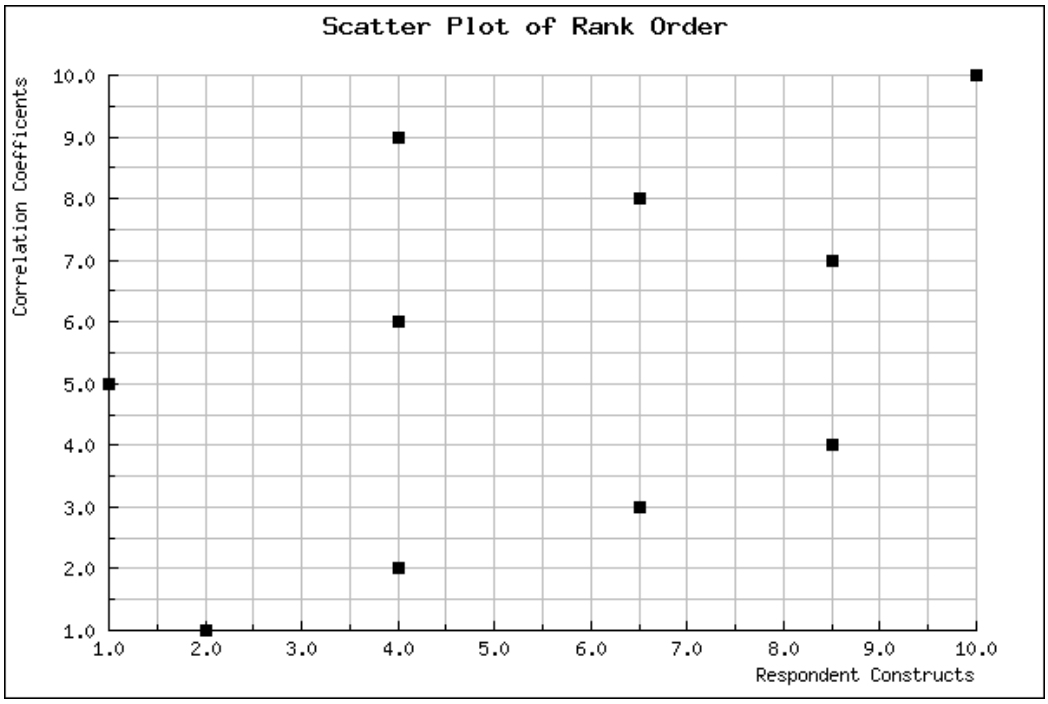


Figure 6.6: rank order correspondence between respondent constructs and correlation coefficients. See the caption of Figure 6.5 for a description of the plot format.

The high correlation coefficient between the two datasets means that there is a reasonable chance that there is a correlation between the characterizations of the respondents' constructs and how much they were thinking in DC and EC terms. The results show this because when the correlation coefficient indicates that a respondent is using a certain amount of combination of DC and EC constructs, the classification of the respondent's constructs will also be about the same amount. For example, the statistics for respondent #6 both ranked it 10, which means the respondent has the highest amount of EC constructs. From this we can say that since respondent #6's constructs correlated with the correlation coefficients, respondent #6 was thinking 100% in EC terms.

Since the two measures are not totally correlated, they might not agree all the time, and thus, the construct percentages would not be exactly correct. For example, based on the correlation coefficients, respondent #5's results are ranked 2 and are most like results generated by the DC dataset. Based on the constructs, respondent #5's results have a ranking of 1 and are 66% DC. The two results are not ranked the same, therefore there may be some error in the 66% rating.

6.2.5 Questionnaire

Overall, the respondents made 21 high, 30 medium, and 29 low novelty ratings. The analysis must determine the effects of the two dependent variables, the question type, DC or EC, and the question weights, high or not high. As described in section 6.1.3, each question was were derived from a match made from either a DC and EC representation of a clock component. The high or not-high classification was determined based on the relative weight of the match used in the question. For example if a particular match produced three gmaps with weights 1.5, 3.4, and 5.5, the questionnaire could use the 1.5 or 3.4 gmaps as the basis for a not-high weight question, and the 5.5 gmap for a high weight question.

First, we expected that EC matches would be more novel because EC can make a wider variety of matches. However, the analysis shows that the respondents thought DC matches were more novel. Figure 6.7 shows 12 of the 21 high novelty ratings were for DC matches.

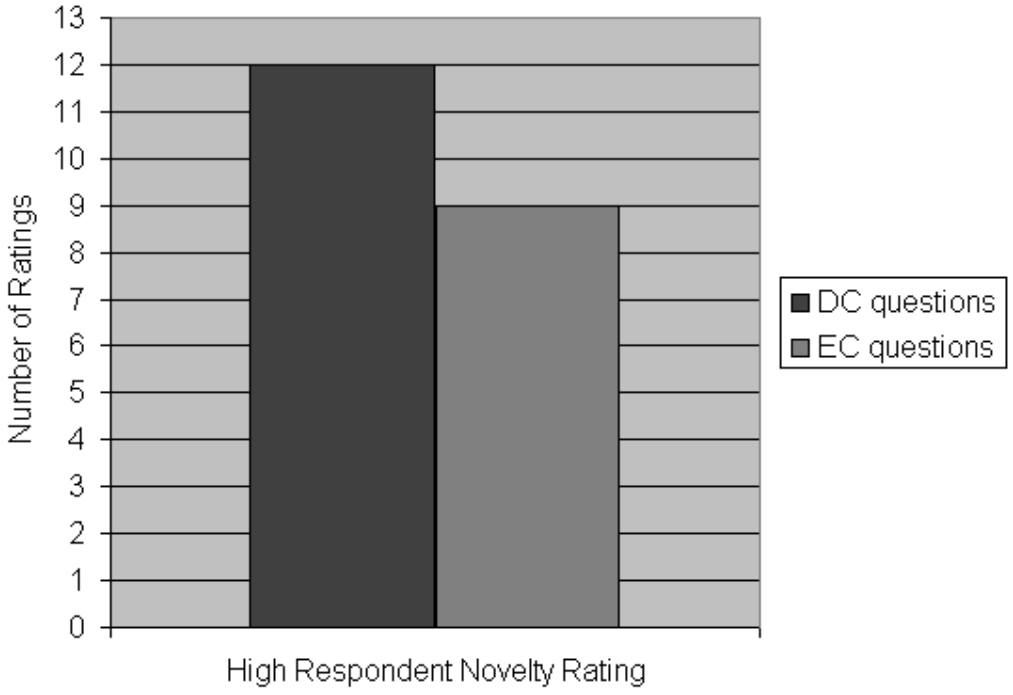


Figure 6.7: the novelty ratings for DC and EC questions

We expected that the lower the SME match weight, the more novel the respondents would rate the match. Since a lower weight means that the match was not a very strong match, we expected that lower weighted matches would seem more novel to the respondents.

The analysis also shows that lower weighted questions are more novel. For the purposes of this analysis, the one mid weighted question was grouped with the low weighted questions to form a "not-high" set of questions. As table 6.6 shows, there were 5 high match weight questions and 3 not-high match weight questions. Nine of 21 high novelty ratings were given to the not-high match weight questions for an average of 3 high novelty ratings per not-high match weight question and 2.4 high novelty ratings per high match weight question.

	high novelty	**medium novelty**	**Low novelty**
high match weight q's	2.4	4.4	3.2
not-high match weight q's	3	2.6	4.3

Table 6.6: average number of novelty ratings per question class

In order to investigate these findings further we performed one additional analysis specifically directed at looking at pairs of questions where only one dependent variable is modified. If the results just presented are true, then the analysis should show the following effects between the pairs of questions in table 6.7.

Questions	**Dependent variable varied**	**Which should be more novel**	**Result**	**Which is more novel based on result**
Q1, Q2	low vs. high weight	Q2	Q2 has one more high novelty rating than Q1	Q2
Q4, Q5	low vs. high weight	Q4	Q4 had 4 high novelty ratings and Q5 has 0	Q4
Q1, Q3	EC vs. DC	Q3	Q3 has one more high novelty rating than Q1	Q3
Q7, Q8	DC vs. EC	Q7	Q7 has one more high novelty rating than Q8	Q7

Table 6.7: shows pairs of questions, the only variable that is modified between them, the resulting tally of high novelty ratings, and which is more novel based on the results. Both column 3 and column 5 should agree.

Thus, as table 6.7 shows, the two observations that DC is more novel than EC and that low weighted questions are more novel than high hold for all four pairs of questions. Since these same clock components were used in each pair, there was only one

dependent variable that was modified, and fewer factors influencing the results. The results provide good support for the truth of the observations, even though the difference for three of the pairs was only one novelty rating.

7. Evaluation of Results

This thesis addresses the problem of determining the impact of using DC, EC, and a combined (called BOTH) knowledge representation. This thesis provides answers to several research questions, and suggests that some hypotheses are true. This section summarizes the results with respect to these research questions and hypotheses. This section also evaluates the process used to generate the results. The original research questions and hypotheses that were proposed in section 1.1 are shown below.

1. Which representation produces more matches?
   EC representations will produce more matches than DC representations. The BOTH representations will produce the most matches.
2. Which representation produces higher weighted matches?
   EC matches will be of lower weight than matches made using representations that are DC. BOTH matches will have the highest weights.
3. Will DC or EC representations produce more novel matches?
   EC representations will produce more novel matches than DC representations.
4. When the level of detail is varied, are the results from questions 1, 2, and 3 still true?
   Yes, the results are not sensitive to the level of detail.
5. How much do matches from each representation correlate with human matches?
   Human matches will correlate best with matches from EC representations.
6. Are human reasons for similarity more DC or EC?
   The humans' reasons will be more EC than DC.

Our computational experiment shows that our hypotheses for questions 1 and 2 are correct for the DC and EC representations, but not for the BOTH representation. EC representations produced higher weighted matches than DC representations, but DC representations produced higher average weighted matches. EC representations had a higher standard deviation than DC representations, which means that EC

representations produce a wider range of results than DC representations. This explains why matches from the EC representation have lower average match weight.

However, the BOTH representation did not create more analogical matches or matches of higher value than the EC or DC representations. Since the number of matches and weight of matches are normalized, much of the BOTH representation is not used in making analogical matches. There could also be some interference happening between the DC and EC representations of devices that is making it harder to make strong matches. The experiments also show that although the matches from the BOTH representation were as varied as EC, there were not as many. The experiments also show that adding extra DC information to EC representations causes them to perform worse than the EC representation alone.

In contrast to our hypothesis for question 3, our analysis shows that both DC and EC representations can produce novel results, not just EC representations. Our results show that DC representations can produce novel results because the respondents in the human experiment rated more DC matches with high novelty than EC matches. They also show that since the respondents rated low valued matches as being more novel, and EC representation produce more low valued matches than DC representation, EC can produce novel results.

Our hypothesis for question 4 was mostly correct. It was not correct for the low detail DC representation because it did not have enough information to produce many matches. Besides the low detail DC dataset, the results show that the same trends occur in the low detail as in the high detail representation.

Our hypothesis was wrong for question 5. Although the human respondent matches were more correlated with EC than DC, they were most correlated with the matches created from the BOTH representations.

Lastly, our hypothesis for question 6 was correct. Our analysis of the human's constructs showed that humans reasons for similarity were more EC than DC.

7.1 Evaluation of Process

There are three areas of our process that worked well, using the functional basis as an ontology, the selection of clock test examples, and the human experiment procedure.

First, using the functional basis as an ontology made developing the test examples

easier. Before discovering the functional basis, we tried to invent terms that were consistent in order to describe the test examples. This proved to be a time consuming task, because when new test examples were added, they often required new terms, which caused us to rethink the definitions of old terms. With the functional basis, this problem occurred less, and the process of writing test examples took less time.

The clock test examples worked well for eliciting analogies about function from the human respondents. When we did earlier experiments with simpler objects like hammers and pens, we noticed the respondents were focusing on surface features. With the clock examples, they had to focus on the functions since the surface features were so different.

Finally, the human experimental procedure was effective at producing interesting results that we could analyze. First, the humans were able to understand the clock test examples, and perform the tasks asked of them. Second, the resulting data was very similar to the computational data, and therefore it allowed for accurate comparisons. In addition, the reasons the respondents provided in the repertory grid allowed us to make conclusions about their reasoning. Thus, not only did this procedure produce numerical data that we could analyze, but it also produced data which we could use to make hypotheses about the respondents' thought processes.

8. Conclusions

The purpose of this thesis was to explore the differences between DC and EC representations of function. To do this we created a knowledge representation and represented a set of clock test examples. We performed a computational experiment with SME and performed an informal human experiment. From these we have discovered some properties of DC and EC representations that may be useful for computer-based design systems and the designers who use them.

First, our experiment shows that computer-based design systems should use EC representations for producing many, potentially novel matches for the designer. This may be useful when the designer is brainstorming. Design systems should use DC representations when the designer is expecting to get fewer matches and wants to find matches that are more relevant to their work. This may occur once the designer has decided on many parts of the design and then just wants to make refinements. Chandrasekaran and Josephson [2000] say that it may be beneficial for designers to switch focuses from EC to DC at a certain point in the design process. This thesis suggests that this decision point may be when the designer wants the design system to change from producing many conceptual designs to producing design refinements.

Second, our experiment shows that a design system could use DC or EC representations to produce novel matches. Unfortunately, our results are inconclusive about whether DC or EC representations are more useful for generating novel matches. On one hand, the low weighted matches that EC representations create can generate novel results. On the other hand, DC representations, which produce few low weighted matches, can also produce novel matches. Thus, more work needs to be done in order to determine which has a greater effect on producing novel matches.

Third, the results show that humans may be thinking more in EC terms than in DC terms. There are multiple pieces of evidence to support this. First, the respondents' similarity reasons are more EC than DC. Second, the humans rated DC matches as being more novel than EC matches. This could mean that the humans are thinking more in EC terms since they find DC matches more surprising and thus more novel.

Fourth, our computational experiments show that although combining DC and EC

representations together does make matches that correlate the best with human matches, the additional work may not be worth it. The results show that adding DC information to the EC representation produces comparatively fewer matches than when the EC representation is used alone. The main observation is that that adding DC information to the EC representation does not add proportionally as many matches. The suggested reason for this is that more interference is occurring.

8.1 Future Work

There are several areas of this thesis that could be expanded in future work. First, the DC and EC representations could be compared using a different analogy system such KDSA [Wolverton and Hayes-Roth, 1995] or AMBR [Kokinov and Petrov, 1988]. These systems use a different, non-symbolic approach to analogy making. Such work would have to define what DC and EC mean in terms of their network representations.

Second, the usefulness of analogical matches created using DC and EC representations could be studied in a complete analogy making system. This thesis only studies the results from the matching step of making analogy. To fully understand the usefulness of the analogical matches, they should be tested in a system that performs all the steps of analogy, which include analog retrieval, analogical inferencing, and learning. Ideally, the resulting analogies should be used for some task. The usefulness of DC versus EC representations should be judged based on their effects on the specific task. As section 2.2 shows, there are several systems that can perform design tasks using analogy. Our system was limited to evaluating analogical matches based on their structural evaluation score. This score may turn out not to be useful in performing a design task such as innovative design by analogy.

Third, the knowledge representation could have been enhanced in two ways. The knowledge representation could be extended to include abstractions. These abstractions could be the ones defined by the functional basis. For example in the functional basis, both import, export, and transfer are different kinds of “channel” and force and torque are both kinds of “energy.” Another kind of abstraction would be to have levels of abstraction for functions and behaviors. It would be interesting to see if our results hold when the level of abstraction is varied.

Another enhancement to the knowledge representation would be to use a pure EC representation. Our EC representation was not purely EC because it did include some information about the device in it. For example, our representation permitted the behavior transmit force from weight to gear. To make this a pure EC behavior, it would need to change to become: transmit force to gear. In this pure EC representation the gear is part of the environment and the behavior is still a behavior for the weight. It would be interesting to compare the results from such a representation to the existing results.

A fourth area of future work is to test these results on a variety of knowledge representations to show that the results are not due to the particular representation used in this thesis or the particular person who created the test examples. Section 2.2 shows that there are several systems that perform analogy. Each of these has a different knowledge representation that could be used to specifically represent DC and EC information.

Fifth, the experimental procedure used in this thesis could be used on a more varied set of test examples in order to verify the results recorded in this thesis. In particular, the test examples could be from in different domains, which have less in common than clock domains, in order to encourage cross-domain analogies. Also, the results could be verified by using a larger set of test examples.

Sixth, there is ambiguity in the human experiment results about whether or not DC or EC representations produce more novel results. There were only ten human respondents in the experiment. If more humans are added to the experiment the results would be more accurate and benefits of one representation versus the other might become more definitive.

Seventh, the respondents' constructs could be analyzed further. In this thesis, the constructs were classified according to the criteria of one researcher. More work in trying to classify the constructs in a different way might provide a different result.

Lastly, there are changes that could be made to the human experiment that would increase the likelihood of finding strong correlations between the human matches and the computer matches. The human experiment shows that the BOTH representation most closely matches the human results. Also, the constructs the humans chose were

about more than just functions and behaviors. These two results hint that adding new types of information to the pure DC or EC representation might make stronger correlations. However, our computational results indicate that adding the wrong information may not increase the results proportionally. Perhaps if the knowledge representation included other primitives such as structure and attributes, the resulting analogical matches would more closely correlate with the human matches because both SME and the human knowledge representations would have more in common.

9. References

Balazs, ME: 1999, *Design Simplification by Analogical Reasoning*. Ph.D. thesis, Worcester Polytechnic Institute, Computer Science Department, Worcester, MA, USA.

Besemer, SP and Treffinger, DJ: 1982, Analysis of Creative Products: Review and Synthesis, *in* GJ Puccio and MC Murdock (eds) *Creativity Assessment: Readings and Resources*, Creative Education Foundation Press, Buffalo, NY, pp. 59-76.

Bhatta, S and Goel, A: 1996, From Design Experiences to Generic Mechanisms: Model-Based Learning in Analogical Design. *Artificial Intelligence in Engineering Design, Analysis and Manufacturing*, **10**:131-136.

Boden, MA: 1994, What is Creativity?, in MA Boden (ed), *Dimensions of Creativity*, The MIT Press, Cambridge, MA, pp. 75-117.

Boden, MA: 2003, Computer Models of Creativity. In Sternberg, R. J., Ed. Handbook of creativity. pp. 351-373.

Brain, M: 2005a, *How Digital Clocks Work*. http://home.howstuffworks.com/digital-clock.htm

Brain, M: 2005b, *How Pendulum Clocks Work*. http://home.howstuffworks.com/clock.htm

Brown, DC: 1992, Design. *Encyclopedia of Artificial Intelligence*. J. Wiley & Sons, 2nd edn. pp. 331-339

Chandrasekaran, B and Josephson, JR: 2000, Function in Device Representation, *Engineering with Computers,* Special Issue on Computer Aided Engineering, **16**: 162-177.

Chandrasekaran, B: 2005 Representing function: Relating functional representation and functional modeling research streams, *AI EDAM*, **19**(2): 65-74.

Falkenhainer, B: 2005, *Structure Mapping Engine Implementation*. http://www.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/reasonng/analogy/sme/0.html

Falkenhainer, B, Forbus, K and Gentner, D: 1989, The structure-mapping engine: Algorithm and examples. *Artificial Intelligence*, **20**(41): 1–63.

French, RM: 2002. The Computational Modeling of Analogy-Making. *Trends in Cognitive Science,* **6**(5), 200-205

Gentner, D, Holyoak, K and Kokinov, B (eds): 2001, *The Analogical Mind: Perspectives from Cognitive Science*, MIT Press.

Gentner, D: 1983, Structure-mapping: A Theoretical Framework for Analogy, *Cognitive Science* **7**(2).

Goel, A: 1997, Design, analogy, and creativity. *IEEE Expert*, **12**(3): 62–70.

Hart, A: 1986, *Knowledge Acquisition for Expert Systems*, McGraw-Hill.

Holyoak, KJ and Thagard P: 1989. Analogical mapping by constraint satisfaction. *Cognitive Science*, **13**:295-355.

Kitamura, Y, Kashiwase, M, Fuse, M and Mizoguchi R: 2004, Deployment of an ontological framework of functional design knowledge, *Journal of Advanced Engineering Informatics*, **18**(2):115-127.

Kokinov, B and Petrov, A: 1988, Associative memory-based reasoning: How to represent and retrieve cases. In *Artificial Intelligence III: Methodology, Systems, and Applications.* 51-58. Amsterdam: Elsevier.

Pahl, G and Beitz, W: 2003 *Engineering design. A systematic approach*. Springer, 2nd edition.

Prabhakar, S and Goal, A: 1996a, Learning about Novel Operating Environments: Designing by Adaptive Modelling, *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing*, **10**:151-156.

Prabhakar, S and Goel, A: 1996b, Functional Modeling of Interactions between Devices and their External Environments for Adaptive Design, *Proc. Modeling and Reasoning about Function workshop*, AAAI-96 Conference, Portland, Oregon, pp. 95-106.

Qian, L and Gero, J: 1992, A Design Support System Using Analogy. In *Proc. of the Second International Conference on AI in Design,* pp. 795-813. Kluwer Academic Publishers.

Qian, L and Gero, J: 1996, Function-behavior-structure paths and their role in analogy-based design, *Artificial Intelligence for Engineering Design, Analysis and Manufacturing,* **10**:289-312.

Rosenman, MA and Gero, JS: 1998, Purpose and function in design: from the socio-cultural to the techno-physical, *Design Studies,* **19**(2): 161-186.

Shafer, G: 1978, *A Mathematical Theory of Evidence*, Princeton University Press, Princeton, New Jersey.

Shaw, LG and Gaines, BR: 2005, *Rep IV 1.10*. http://repgrid.com, Cobble Hill, BC, Canada.

Stone, RB and Wood, KL: 1999, Development of a functional basis for design. *Proc. ASME Design Theory and Methodology Conference*, DETC99/DTM-8765. pp. 359-370.

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

Vermaas, PE: 2002, A Critical Analysis of John Gero's Function-Behavior-Structure Model of Designing. In *Proceedings of the Design Research Society International Conference at Brunel University*, 1163-1172.

Wolverton, M and Hayes-Roth, B: 1995, Finding analogues for innovative design. In *Third Int'l Conference on Computational Models of Creative Design*.

10. Appendix

Appendix A Computational Experiment

A.1 Device Input Format

This appendix shows the input file used to describe the pendulum and digital clocks. Also, this appendix describes how we encoded the part-of mapping. The file is in an XML (http://www.w3.org/XML/) format. It also uses abbreviations to make it succinct. They are listed below:

Abbreviation	Meaning
t	type
d	DC
e	EC
n	name
beh	behavior
subfun	subfunction
sX	a reference to a subfunction with id X
f	function
t	type
ent	entity
d1	device1
d2	device2
i	unique id
<!--	beginning of a comment
-->	end of a comment

Table of the abbreviations used in the input files

Pendulum clock:

```
<device n="pendulumclock">

<!-- get torque to the escapement -->
<beh t="e" i="1" n="transmit" ent="force" d1="earth" d2="weight"/>
<beh t="e"  i="2" n="transmit" ent="force" d1="weight" d2="weightgear"/>
<beh t="e"  i="3" n="transmit" ent="torque" d1="weightgear" d2="escapementgear"/>
<beh t="e"  i="20" n="transmit" ent="force" d1="earth" d2="pendulum"/>
<beh t="e"  i="21" n="transmit" ent="torque" d1="pendulum" d2="stopper"/>
<beh t="e"  i="22" n="decouple" d1="stopper" d2="escapement"/>
<subfun t="e" i ="1" beh="1,2,20,21,22,3" goal="get torque to the escapement gear"/>

<beh t="e"  i="4" n="transmit" ent="torque" d1="escapementgear" d2="inputgear"/>
<beh t="e"  i="5" n="transmit" ent="torque" d1="inputgear" d2="secondgear"/>
```

```
    <beh t="e"  i="6" n="transmit" ent="torque" d1="secondgear" d2="minutegear"/>
    <subfun t="e" i="2" beh="s1,4,5,6" goal="the torque gets to the second gear"/>

    <beh t="e"  i="7" n="transmit" ent="torque" d1="secondgear" d2="secondhand"/>
    <subfun t="e" i="3" beh="s2,7" goal="turn second hand" />

    <beh t="e"  i="8" n="transmit" ent="torque" d1="secondgear" d2="minutegear"/>
    <beh t="e"  i="9" n="transmit" ent="torque" d1="minutegear" d2="minutehand"/>
    <subfun t="e" i="4" beh="s2,8,9" goal="turn the minute hand" />

    <!-- reset function
    when the gear release lever is enabled, the escapement transmits directly to the second
gear -->
    <beh t="e"  i="12" n="transmit" ent="force" d1="human" d2="gear-release-lever"/>
    <beh t="e"  i="10" n="transmit" ent="torque" d1="escapementgear" d2="secondgear"/>
    <beh t="e"  i="11" n="change-between" ent="torque" d1="secondgear" d2="minutegear"/>
    <f t="e" i="30" beh="s1,12,10,11,9,34" goal="turn the minute hand faster and human sees
it" />
    <f t="e" i="40" beh="s1,12,10,7,32" goal="turn the second hand faster and human sees it"
/>

    <!-- human sees it -->
    <beh t="e"  i="32" n="transmit" ent="visual-signal" d1="secondhand" d2="human"/>
    <f t="e" i="50" beh="s3,32" goal="human sees secondhand" />

    <beh t="e"  i="34" n="transmit" ent="visual-signal" d1="minutehand" d2="human"/>
    <f t="e" i="60" beh="s4,34" goal="human sees minutehand" />

    <!-- get torque to the escapement -->
    <beh t="d"  i="1" n="import" ent="force" d1="weight"/>
    <beh t="d"  i="2" n="export" ent="force" d1="weight"/>
    <beh t="d"  i="3" n="import" ent="force" d1="weightgear"/>
    <beh t="d"  i="4" n="export" ent="torque" d1="weightgear"/>
    <beh t="d"  i="5" n="import" ent="torque" d1="escapementgear"/>
    <beh t="d"  i="6" n="export" ent="torque" d1="escapementgear"/>
    <subfun t="d" i ="10" beh="1,2,3,4,5,6,s36" goal="get torque to the escapement gear"/>

    <beh t="d"  i="7" n="import" ent="torque" d1="inputgear"/>
    <beh t="d"  i="8" n="export" ent="torque" d1="inputgear"/>
    <beh t="d"  i="9" n="import" ent="torque" d1="secondgear"/>
    <beh t="d"  i="10" n="export" ent="torque" d1="secondgear"/>
    <subfun t="d" i="12" beh="s10,7,8,9,10" goal="torque to second gear"/>

    <beh t="d"  i="23" n="import" ent="torque" d1="secondhand"/>
    <beh t="d"  i="24" n="rotate" d1="secondhand"/>
    <f t="d" i="1" beh="s12, 23, 24" goal = "show 1 second increment"/>

    <beh t="d"  i="11" n="import" ent="torque" d1="minutegear"/>
    <beh t="d"  i="12" n="export" ent="torque" d1="minutegear"/>
    <beh t="d"  i="21" n="import" ent="torque" d1="minutehand"/>
    <beh t="d"  i="22" n="rotate" d1="minutehand"/>
    <f t="d" i="2" beh="s12, 11, 12, 21, 22" goal = "show 1 min increment"/>

    <beh t="d"  i="13" n="import" ent="torque" d1="stopper"/>
    <beh t="d"  i="14" n="export" ent="force" d1="stopper"/>
    <beh t="d"  i="15" n="import" ent="force" d1="pendulum"/>
    <beh t="d"  i="16" n="export" ent="torque" d1="pendulum"/>
    <subfun t="d" i="36" beh="13,14,15,16" goal="actuate the pendulum"/>
```

```
<beh t="d"  i="17" n="import" ent="force" d1="gear-release-lever"/>
<beh t="d"  i="18" n="stop" ent="force" d1="gear-release-lever"/>
<beh t="d"  i="19" n="change" ent="torque" d1="secondgear"/>
<beh t="d"  i="20" n="change" ent="torque" d1="minutegear"/>
<!-- don't record change torque on the sec and min hands -->
<f t="d" i="300" beh="s10,17,18,19,23,24" goal="set second hand"/>
<f t="d" i="400" beh="s10,17,18,19,20,21,22" goal="set minute hand"/>
</device>
```

Digital clock:

```
<device n="digitalclock">
<!-- this is a clock that has a two digit seconds display -->

<beh t="d" i="1" n="import" ent="eforce" d1="plug"/>
<beh t="d" i="2" n="export" ent="eforce" d1="plug"/>
<beh t="d" i="3" n="import" ent="eforce" d1="bridge"/>
<beh t="d" i="4" n="export" ent="eforce" d1="bridge"/>
<beh t="d" i="5" n="import" ent="eforce" d1="diode"/>
<beh t="d" i="6" n="change" ent="eforce" d1="diode"/>

<beh t="e" i="1" n="couple" d1="plug" d2="wall"/>
<beh t="e" i="2" n="transmit" ent="eforce" d1="plug" d2="bridge"/>
<beh t="e" i="3" n="transmit" ent="eforce" d1="bridge" d2="diode"/>
<beh t="e" i="4" n="change-between" ent="eforce" d1="diode" d2="tbdiv10"/>

<!-- invented a new term called change-between which is different from change in that
change-between specified two devices instead of one. This was done so I could make an EC
version of change -->
<!-- this is the power provider functionality -->
<subfun t="d" i ="1" beh="1,2,3,4,5,6" goal="eforce out of diode"/>
<subfun t="e" i ="10" beh="1,2,3,4" goal="eforce to tbdiv10"/>

<!-- time base functionality -->

<beh t="d" i="7" n="import" ent="eforce" d1="tbdiv10"/>
<beh t="d" i="8" n="export" ent="signal" d1="tbdiv10"/>
<beh t="d" i="9" n="import" ent="signal" d1="tbdiv6"/>
<beh t="d" i="10" n="export" ent="signal" d1="tbdiv6"/>

<beh t="e" i="5" n="transmit" ent="signal" d1="tbdiv10" d2="tbdiv6"/>
<beh t="e" i="6" n="transmit" ent="signal" d1="tbdiv6" d2="div10"/>

<subfun t="d" i ="2" beh="s1,7,8,9,10" goal="eforce out of tbdiv6"/>
<subfun t="e" i ="12" beh="s10,5,6" goal="eforce to div10"/>

<!-- gear functionality -->

<!-- import signal to in and to the reset area.. what can I call that, change? -->
<beh t="d" i="11" n="import" ent="signal" d1="div10"/>
<!-- output it to bc and to out -->
<beh t="d" i="12" n="export" ent="signal" d1="div10"/>
<!-- could call this change outward, there's not distinction of in or out or which parts -->
<beh t="d" i="13" n="change" ent="signal" d1="div10"/>
<beh t="d" i="14" n="import" ent="signal" d1="div6"/>
<beh t="d" i="15" n="export" ent="signal" d1="div6"/>
```

```
    <beh t="d" i="16" n="change" ent="signal" d1="div6"/>

    <!--<beh t="e" i="7" n="transmit" ent="signal" d1="switch" d2="div10"/>-->
    <beh t="e" i="8" n="transmit" ent="signal" d1="div10" d2="div6"/>
    <beh t="e" i="9" n="transmit" ent="signal" d1="div10" d2="bc10"/>
    <!-- signal that the div6 should reset by changing the signal -->
    <beh t="e" i="10" n="change-between" ent="signal" d1="div10" d2="div6"/>
    <!-- here change means give a special signal for resetting the bc -->
    <beh t="e" i="11" n="change-between" ent="signal" d1="div10" d2="bc10"/>
    <beh t="e" i="12" n="change-between" ent="signal" d1="div6" d2="bc6"/>
    <beh t="e" i="13" n="transmit" ent="signal" d1="div6" d2="bc6"/>

    <subfun t="d" i ="3" beh="s2,11,12,14,15,25,27,21,22" goal="increment the sec tens"/>
    <subfun t="d" i ="4" beh="s7, s2,11,13,16,26,21,22" goal="reset the minutehand"/>
    <subfun t="d" i ="5" beh="s2,11,12,28,30,19,20" goal="increment the sec ones"/>
    <subfun t="d" i ="6" beh="s7, s2,11,13,29,19,20" goal="reset the secondhand"/>

    <!-- bc's -->

    <beh t="d" i="25" n="import" ent="signal" d1="bc6"/>
    <beh t="d" i="26" n="change" ent="signal" d1="bc6"/>
    <beh t="d" i="27" n="export" ent="signal" d1="bc6"/>
    <beh t="d" i="28" n="import" ent="signal" d1="bc10"/>
    <beh t="d" i="29" n="change" ent="signal" d1="bc10"/>
    <beh t="d" i="30" n="export" ent="signal" d1="bc10"/>

    <beh t="e" i="32" n="transmit" ent="signal" d1="bc6" d2="secTensDisplay"/>
    <beh t="e" i="33" n="transmit" ent="signal" d1="bc10" d2="secOnesDisplay"/>
    <!-- leave out position -->
    <beh t="e" i="16" n="transmit" ent="visual-signal" d1="secOnesDisplay" d2="human"/>
    <beh t="e" i="17" n="transmit" ent="visual-signal" d1="secTensDisplay" d2="human"/>

    <subfun t="e" i ="13" beh="s12,8,13,32,17" goal="signal to bc6 increment the sec tens"/>
    <subfun t="e" i ="14" beh="s12,s17,10,12,32,17" goal="changed to bc6 reset the sec
tens"/>
    <subfun t="e" i ="15" beh="s12,9,33,16" goal="signal to bc10 increment the sec ones"/>
    <subfun t="e" i ="16" beh="s12,s17,11,33,16" goal="reset the sec ones"/>

    <!-- face -->
    <beh t="d" i="19" n="import" ent="signal" d1="secOnesDisplay"/>
    <beh t="d" i="20" n="display" ent="visual-signal" d1="secOnesDisplay"/>
    <beh t="d" i="21" n="import" ent="signal" d1="secTensDisplay"/>
    <beh t="d" i="22" n="display" ent="visual-signal" d1="secTensDisplay"/>

    <!-- switch -->
    <beh t="d" i="17" n="import" ent="force" d1="switch"/>
    <beh t="d" i="18" n="export" ent="signal" d1="switch"/>
    <beh t="e" i="14" n="transmit" ent="force" d1="human" d2="switch"/>
    <beh t="e" i="15" n="transmit" ent="signal" d1="switch" d2="div10"/>
    <subfun t="d" i ="7" beh="17,18" goal="hit the switch"/>
    <subfun t="e" i ="17" beh="15,14" goal="hit the switch"/>

    <!-- functions -->
    <f t="d" i ="100" beh="s3" goal="increment the sec tens"/>
    <f t="d" i ="200" beh="s4" goal="reset the minutehand"/>
    <f t="d" i ="300" beh="s5" goal="increment the sec ones"/>
    <f t="d" i ="400" beh="s6" goal="reset the secondhand"/>
    <f t="e" i ="1000" beh="s13" goal="increment the sec tens"/>
```

```
<f t="e" i ="2000" beh="s14" goal="reset the sec tens"/>
<f t="e" i ="3000" beh="s15" goal="increment the sec ones"/>
<f t="e" i ="4000" beh="s16" goal="reset the sec ones"/>

</device>
```

Part-of mapping:

The part-of mapping information is how the test harness code knows which clock subcomponents are part of which clock components. For example, both the secondhand and the minutehand are part of the pendulum face. We hard coded the part of mapping information in java code so it is not an actual input file. The tables below show the part of mapping information.

Device	**Part-of**
weightgear	pendulumpower
weight	pendulumpower
escapementgear	pendulumgear
inputgear	pendulumtimebase
secondgear	pendulumtimebase
minutegear	pendulumgear
gear-release-lever	pendulumgear
secondhand	pendulumface
minutehand	pendulumface

Part-of mapping for the pendulum clock

Device	**Part-of**
plug	digitalpowerprovider
bridge	digitalpowerprovider
diode	digitalpowerprovider
tbdiv10	digitaltimebase
tbdiv6	digitaltimebase
switch	digitalgear
div10	digitalgear
div6	digitalgear
bc10	digitalgear
bc6	digitalgear
secOnesDisplay	digitalface
setTensDisplay	digitalface

Part-of mapping for the digital clock

A.2 SME Input Format

This appendix shows the DC and EC versions of the SME input format used for the bridge subcomponent and also the EC version of the diode subcomponent.

Bridge DC:

```
    (sme:defEntity bridge :type inanimate)
    (sme:defDescription bridge_DC
     entities (bridge)
     expressions (
    ((import eforce bridge ) :name *import_eforce_bridge)
    ((behavior *import_eforce_bridge) :name *behavior_import_eforce_bridge)
    ((export eforce bridge ) :name *export_eforce_bridge)
    ((behavior *export_eforce_bridge) :name *behavior_export_eforce_bridge)
    ((behavior-set  *behavior_import_eforce_bridge *behavior_export_eforce_bridge )
     :name *behavior_set_behavior_import_eforce_bridge_behavior_export_eforce_bridge)
    ((DC *behavior_set_behavior_import_eforce_bridge_behavior_export_eforce_bridge)
:name   *function_behavior_import_eforce_bridge_behavior_export_eforce_bridge)))
```

Bridge EC:

```
    (sme:defEntity plug :type inanimate)
    (sme:defEntity bridge :type inanimate)
    (sme:defEntity diode :type inanimate)
    (sme:defDescription bridge_EC
     entities (plug bridge diode )
     expressions (
    ((transmit eforce plug bridge ) :name *transmit_eforce_plug_bridge)
    ((behavior *transmit_eforce_plug_bridge) :name *behavior_transmit_eforce_plug_bridge)
    ((transmit eforce bridge diode ) :name *transmit_eforce_bridge_diode)
    ((behavior *transmit_eforce_bridge_diode) :name
*behavior_transmit_eforce_bridge_diode)
    ((behavior-set  *behavior_transmit_eforce_plug_bridge
*behavior_transmit_eforce_bridge_diode ) :name
*behavior_set_behavior_transmit_eforce_plug_bridge_behavior_transmit_eforce_bridge_diod
e)
    ((EC
*behavior_set_behavior_transmit_eforce_plug_bridge_behavior_transmit_eforce_bridge_diod
e) :name
*function_behavior_transmit_eforce_plug_bridge_behavior_transmit_eforce_bridge_diode)))
```

Diode EC:

```
    (sme:defEntity bridge :type inanimate)
    (sme:defEntity diode :type inanimate)
    (sme:defEntity tbdiv10 :type inanimate)
    (sme:defDescription diode_EC
    entities (bridge diode tbdiv10 )
    expressions (
    ((transmit eforce bridge diode ) :name *transmit_eforce_bridge_diode)
    ((behavior *transmit_eforce_bridge_diode) :name
*behavior_transmit_eforce_bridge_diode)
    ((change-between eforce diode tbdiv10 ) :name *change-between_eforce_diode_tbdiv10)
```

```
    ((behavior *change-between_eforce_diode_tbdiv10) :name *behavior_change-
between_eforce_diode_tbdiv10)
    ((behavior-set  *behavior_transmit_eforce_bridge_diode *behavior_change-
between_eforce_diode_tbdiv10 )
     :name *behavior_set_behavior_transmit_eforce_bridge_diode_behavior_change-
between_eforce_diode_tbdiv10)
    ((EC *behavior_set_behavior_transmit_eforce_bridge_diode_behavior_change-
between_eforce_diode_tbdiv10) :name
*function_behavior_transmit_eforce_bridge_diode_behavior_change-
between_eforce_diode_tbdiv10)))
```

A.3 SME Example Raw Output

This appendix shows the SME output for the comparison between the EC versions of the bridge and bc10 subcomponents.

```
            SME Version 2E
       Analogical Match from BRIDGE_EC to BC10_EC.

    Rule File: true-analogy.rules
    ------------------------------------------------------------------------------------------
    # MH's | # Gmaps |      1st,2nd,Worst    |  STD  | Merge Step 3 |   CI   | RelGroups |  1-1 |
      22  |     3   |  9.01 /  4.46 /  4.41 |  0.00 |   ACTIVE     | ACTIVE |     OFF    | FULL |
    ------------------------------------------------------------------------------------------
    Total Run Time:    0 Minutes,  0.030 Seconds
    BMS Run Time:      0 Minutes,  0.030 Seconds
    Best Gmaps: { 3 }

    Match Hypotheses:
        (0.7582  0.0000)  (PLUG DIV10)
        (0.9488  0.0000)  (*TRANSMIT_EFORCE_PLUG_BRIDGE
*TRANSMIT_SIGNAL_DIV10_BC10)
        (0.7626  0.0000)  (BRIDGE SECONESDISPLAY)
        (0.7626  0.0000)  (PLUG BC10)
        (0.9556  0.0000)  (*TRANSMIT_EFORCE_PLUG_BRIDGE
*TRANSMIT_SIGNAL_BC10_SECONESDISPLAY)
        (0.9450  0.0000)  (*BEHAVIOR_TRANSMIT_EFORCE_PLUG_BRIDGE
*BEHAVIOR_TRANSMIT_SIGNAL_DIV10_BC10)
        (0.6647  0.2682)  (*BEHAVIOR_TRANSMIT_EFORCE_PLUG_BRIDGE
*BEHAVIOR_CHANGE-BETWEEN_SIGNAL_DIV10_BC10)
        (0.9856  0.0000)  (*BEHAVIOR_TRANSMIT_EFORCE_PLUG_BRIDGE
*BEHAVIOR_TRANSMIT_SIGNAL_BC10_SECONESDISPLAY)
        (0.7582  0.0000)  (DIODE BC10)
        (0.7582  0.0000)  (BRIDGE DIV10)
        (0.9488  0.0000)  (*TRANSMIT_EFORCE_BRIDGE_DIODE
*TRANSMIT_SIGNAL_DIV10_BC10)
        (0.7626  0.0000)  (DIODE SECONESDISPLAY)
        (0.9426  0.0000)  (BRIDGE BC10)
        (0.9967  0.0000)  (EFORCE SIGNAL)
        (0.9556  0.0000)  (*TRANSMIT_EFORCE_BRIDGE_DIODE
*TRANSMIT_SIGNAL_BC10_SECONESDISPLAY)
        (0.9450  0.0000)  (*BEHAVIOR_TRANSMIT_EFORCE_BRIDGE_DIODE
*BEHAVIOR_TRANSMIT_SIGNAL_DIV10_BC10)
        (0.6647  0.2682)  (*BEHAVIOR_TRANSMIT_EFORCE_BRIDGE_DIODE
*BEHAVIOR_CHANGE-BETWEEN_SIGNAL_DIV10_BC10)
        (0.9856  0.0000)  (*BEHAVIOR_TRANSMIT_EFORCE_BRIDGE_DIODE
*BEHAVIOR_TRANSMIT_SIGNAL_BC10_SECONESDISPLAY)
```

(0.9227 0.0000)
(*BEHAVIOR_SET_BEHAVIOR_TRANSMIT_EFORCE_PLUG_BRIDGE_BEHAVIOR_TRAN
SMIT_EFORCE_BRIDGE_DIODE
*BEHAVIOR_SET_BEHAVIOR_TRANSMIT_SIGNAL_DIV10_BC10_BEHAVIOR_TRANSMIT
_SIGNAL_BC10_SECONESDISPLAY)
(0.9227 0.0000)
(*BEHAVIOR_SET_BEHAVIOR_TRANSMIT_EFORCE_PLUG_BRIDGE_BEHAVIOR_TRAN
SMIT_EFORCE_BRIDGE_DIODE *BEHAVIOR_SET_BEHAVIOR_CHANGE-
BETWEEN_SIGNAL_DIV10_BC10_BEHAVIOR_TRANSMIT_SIGNAL_BC10_SECONESDIS
PLAY)
(0.7900 0.0000)
(*FUNCTION_BEHAVIOR_TRANSMIT_EFORCE_PLUG_BRIDGE_BEHAVIOR_TRANSMIT_
EFORCE_BRIDGE_DIODE
*FUNCTION_BEHAVIOR_TRANSMIT_SIGNAL_DIV10_BC10_BEHAVIOR_TRANSMIT_SIG
NAL_BC10_SECONESDISPLAY)
(0.7900 0.0000)
(*FUNCTION_BEHAVIOR_TRANSMIT_EFORCE_PLUG_BRIDGE_BEHAVIOR_TRANSMIT_
EFORCE_BRIDGE_DIODE *FUNCTION_BEHAVIOR_CHANGE-
BETWEEN_SIGNAL_DIV10_BC10_BEHAVIOR_TRANSMIT_SIGNAL_BC10_SECONESDIS
PLAY)

Gmap #1: (EFORCE SIGNAL) (PLUG BC10) (BRIDGE SECONESDISPLAY)
(*TRANSMIT_EFORCE_PLUG_BRIDGE
*TRANSMIT_SIGNAL_BC10_SECONESDISPLAY)
(*BEHAVIOR_TRANSMIT_EFORCE_PLUG_BRIDGE
*BEHAVIOR_TRANSMIT_SIGNAL_BC10_SECONESDISPLAY)
Emaps: (EFORCE SIGNAL) (PLUG BC10) (BRIDGE SECONESDISPLAY)
Weight: 4.4630
|| # MH's: 5 || # Emaps: 3 || Max/Ave Order: 2/0.60 || Predicate Orders: (3 1 1) ||
Candidate Inferences:

Gmap #2: (EFORCE SIGNAL) (BRIDGE DIV10) (DIODE BC10)
(*TRANSMIT_EFORCE_BRIDGE_DIODE *TRANSMIT_SIGNAL_DIV10_BC10)
(*BEHAVIOR_TRANSMIT_EFORCE_BRIDGE_DIODE
*BEHAVIOR_TRANSMIT_SIGNAL_DIV10_BC10)
Emaps: (EFORCE SIGNAL) (BRIDGE DIV10) (DIODE BC10)
Weight: 4.4069
|| # MH's: 5 || # Emaps: 3 || Max/Ave Order: 2/0.60 || Predicate Orders: (3 1 1) ||
Candidate Inferences:

Gmap #3: (DIODE SECONESDISPLAY) (*TRANSMIT_EFORCE_BRIDGE_DIODE
*TRANSMIT_SIGNAL_BC10_SECONESDISPLAY)
(*BEHAVIOR_TRANSMIT_EFORCE_BRIDGE_DIODE
*BEHAVIOR_TRANSMIT_SIGNAL_BC10_SECONESDISPLAY)
(EFORCE SIGNAL) (PLUG DIV10) (BRIDGE BC10)
(*TRANSMIT_EFORCE_PLUG_BRIDGE *TRANSMIT_SIGNAL_DIV10_BC10)
(*BEHAVIOR_TRANSMIT_EFORCE_PLUG_BRIDGE
*BEHAVIOR_TRANSMIT_SIGNAL_DIV10_BC10)
(*BEHAVIOR_SET_BEHAVIOR_TRANSMIT_EFORCE_PLUG_BRIDGE_BEHAVIOR_TRAN
SMIT_EFORCE_BRIDGE_DIODE
*BEHAVIOR_SET_BEHAVIOR_TRANSMIT_SIGNAL_DIV10_BC10_BEHAVIOR_TRANSMIT
_SIGNAL_BC10_SECONESDISPLAY)

(*FUNCTION_BEHAVIOR_TRANSMIT_EFORCE_PLUG_BRIDGE_BEHAVIOR_TRANSMIT_
EFORCE_BRIDGE_DIODE
*FUNCTION_BEHAVIOR_TRANSMIT_SIGNAL_DIV10_BC10_BEHAVIOR_TRANSMIT_SIG

NAL_BC10_SECONESDISPLAY)
Emaps: (DIODE SECONESDISPLAY) (EFORCE SIGNAL) (PLUG DIV10) (BRIDGE BC10)
Weight: 9.0078
|| # MH's: 10 || # Emaps: 4 || Max/Ave Order: 4/1.30 || Predicate Orders: (4 2 2 1 1) ||
Candidate Inferences: { }

A.4 Computational Experiment Raw Data

This appendix contains raw data and summary statistics of the data used in the computational experiment for both the high and low detail datasets

High complexity data: (note: our SME implementation was not able to compute matches involving the high detail version of the BOTH representation for the digitalgear. Any affected data is be marked as N/A.)

num gmaps	**average**	**stdev**	**95% confidence interval for average**
DC	1.238946	0.322368	0.969 to 1.508
EC	2.566369	0.755572	1.935 to 3.198
BOTH	1.988946	0.4136	1.606 to 2.371
average stdev			
DC	0.043522	0.030191	0.018 to 0.069
EC	0.121214	0.039503	0.088 to 0.154
BOTH	0.127673	0.043779	0.087 to 0.168
average gmap weight			
DC	0.55795	0.18524	0.403 to 0.713
EC	0.470516	0.128346	0.363 to 0.578
BOTH	0.396742	0.073461	0.329 to 0.465
max gmap weight			
DC	0.608569	0.203405	0.439 to 0.779
EC	0.646049	0.158451	0.514 to 0.779
BOTH	0.623298	0.149767	0.485 to 0.762

Summary statistics for high detail dataset

	DC	**EC**	**BOTH**
pendulumpower	1.142857	3.571429	1.809524
pendulumgear	1.214286	2.369048	1.97619
pendulumtimebase	1.380952	2.047619	1.916667
pendulumface	0.952381	2.619048	2.547619
digitalpower	1.959184	1.571429	1.583333
digitalgear	1.166667	3.733333	N/A
digitaltimebase	1.142857	2	1.541667
digitalface	0.952381	2.619048	2.547619

Average number of gmaps for high detail dataset

	DC	EC	BOTH
pendulumpower	0.04886	0.113636	0.179519
pendulumgear	0.063792	0.083575	0.072573
pendulumtimebase	0.101703	0.165139	0.170464
pendulumface	0.01023	0.147524	0.105597
digitalpower	0.03669	0.125547	0.091527
digitalgear	0.028685	0.045594	N/A
digitaltimebase	0.048667	0.141171	0.168447
digitalface	0.009545	0.147524	0.105585

Average standard deviation of gmaps for high detail dataset

	DC	EC	BOTH
pendulumpower	0.769669	0.642602	0.488142
pendulumgear	0.414677	0.300454	0.265118
pendulumtimebase	0.696866	0.48652	0.456867
pendulumface	0.47522	0.550476	0.372195
digitalpower	0.602276	0.444174	0.384429
digitalgear	0.273718	0.269124	N/A
digitaltimebase	0.78779	0.520303	0.438291
digitalface	0.443385	0.550476	0.372151

Average of average gmap weight for high detail dataset

	DC	EC	BOTH
pendulumpower	0.829464	0.843419	0.841239
pendulumgear	0.477087	0.461967	0.418173
pendulumtimebase	0.809779	0.75296	0.733953
pendulumface	0.485384	0.720602	0.542862
digitalpower	0.647804	0.60006	0.54428
digitalgear	0.317759	0.371128	N/A
digitaltimebase	0.848406	0.69765	0.739781
digitalface	0.452868	0.720602	0.542797

Average highest gmap weight for high detail dataset

Low detail dataset:

num gmaps	average	stdev	95% confidence interval for average
DC	0.295238	0.061721	0.269 to 0.322
EC	0.942143	0.236014	0.840 to 1.044
BOTH	0.488306	0.099092	0.445 to 0.531
average stdev			
DC	0	0	0.000 to 0
EC	0.179626	0.092046	0.140 to 0.219
BOTH	0.251235	0.071771	0.220 to 0.282
average gmap weight			
DC	0.690712	0.152631	0.625 to 0.757
EC	0.554337	0.156061	0.487 to 0.622
BOTH	0.438991	0.083243	0.403 to 0.475
max gmap weight			
DC	0.690712	0.152631	0.625 to 0.757
EC	0.762853	0.1965	0.678 to 0.848
BOTH	0.708088	0.135355	0.650 to 0.767

Summary statistics for low detail dataset

	DC	EC	BOTH
weightgear	0.333333333	1.116667	0.558333
weight	0.333333333	1.116667	0.558333
escapementgear	0.333333333	1.03	0.64375
inputgear	0.333333333	1.116667	0.558333
gear-release-lever	0.333333333	1.375	0.57
secondgear	0.2	0.615	0.41
minutegear	0.2	0.65	0.379167
secondhand	0.333333333	1.116667	0.558333
minutehand	0.333333333	1.116667	0.558333
plug	0.333333333	0.7	0.366667
bridge	0.333333333	1.116667	0.558333
diode	0.333333333	0.816667	0.425
tbdiv10	0.333333333	0.783333	0.408333
tbdiv6	0.333333333	1.116667	0.558333
switch	0.333333333	1.116667	0.558333
div10	0.2	0.705	0.47
div6	0.2	0.583333	0.318182
bc10	0.2	0.68	0.34
bc6	0.2	0.68	0.34
secOnesDisplay	0.333333333	1.116667	0.558333
secTensDisplay	0.333333333	1.116667	0.558333

Average number of gmaps for low detail dataset

	DC	EC	BOTH
weightgear	0	0.260253	0.340156
weight	0	0.248352	0.337913
escapementgear	0	0.168742	0.247488
inputgear	0	0.248352	0.337913
gear-release-lever	0	0.001442	0.114336
secondgear	0	0.104799	0.16616
minutegear	0	0.153257	0.211404
secondhand	0	0.260253	0.252031
minutehand	0	0.260253	0.252031
plug	0	0.001088	0.209258
bridge	0	0.248352	0.337913
diode	0	0.035495	0.12888
tbdiv10	0	0.073081	0.24799
tbdiv6	0	0.248352	0.337913
switch	0	0.260253	0.340156
div10	0	0.091467	0.141004
div6	0	0.172643	0.23946
bc10	0	0.207607	0.267592
bc6	0	0.207607	0.267592
secOnesDisplay	0	0.260253	0.249374
secTensDisplay	0	0.260253	0.249374

Average standard deviation for low detail dataset

	DC	EC	BOTH
weightgear	0.828887	0.613551	0.496739
weight	0.850575	0.643483	0.518353
escapementgear	0.850575	0.460551	0.430025
inputgear	0.850575	0.643483	0.518353
gear-release-lever	0.462254	1.024629	0.58609
secondgear	0.668185	0.285384	0.266148
minutegear	0.668185	0.466286	0.401762
secondhand	0.500347	0.613551	0.441198
minutehand	0.500347	0.613551	0.441198
plug	0.850575	0.494745	0.477985
bridge	0.850575	0.643483	0.518353
diode	0.565652	0.513579	0.400072
tbdiv10	0.828887	0.503121	0.476417
tbdiv6	0.850575	0.643483	0.518353
switch	0.828887	0.613551	0.496739
div10	0.63043	0.260022	0.256575
div6	0.668185	0.417643	0.346935
bc10	0.668185	0.479936	0.377618
bc6	0.668185	0.479936	0.377618
secOnesDisplay	0.457441	0.613551	0.436143
secTensDisplay	0.457441	0.613551	0.436143

Average of average gmap weight for low detail dataset

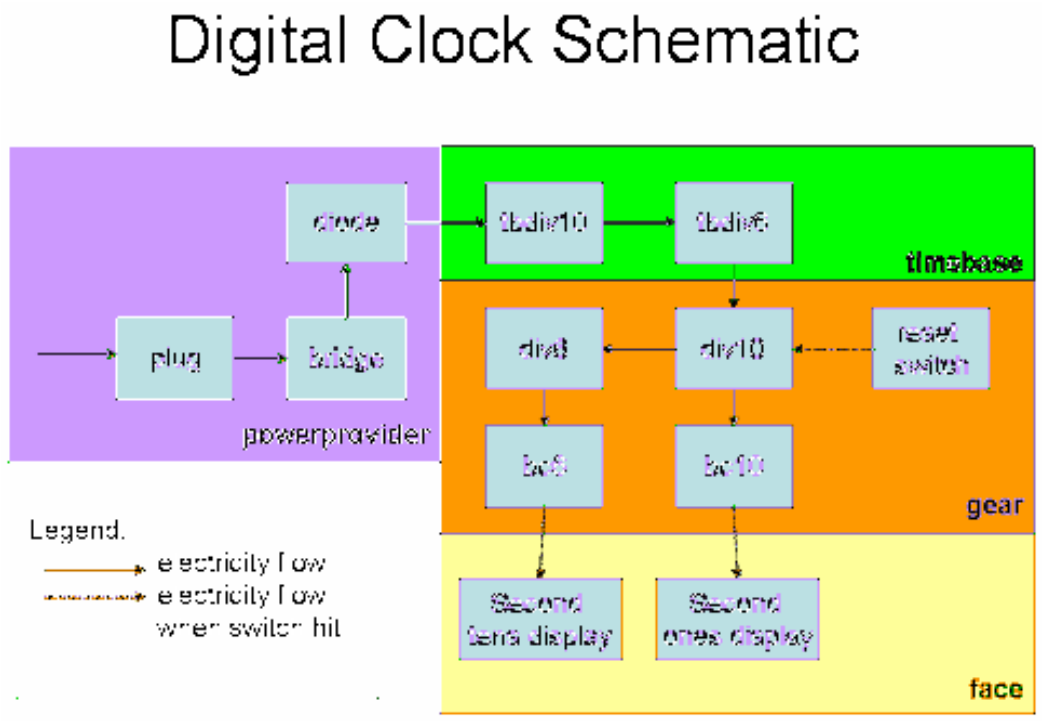
	DC	EC	BOTH
weightgear	0.828887	0.914068	0.868377
weight	0.850575	0.930248	0.888321
escapementgear	0.850575	0.662219	0.706636
inputgear	0.850575	0.930248	0.888321
gear-release-lever	0.462254	1.025675	0.673262
secondgear	0.668185	0.422692	0.47807
minutegear	0.668185	0.661599	0.639624
secondhand	0.500347	0.914068	0.721154
minutehand	0.500347	0.914068	0.721154
plug	0.850575	0.495539	0.627994
bridge	0.850575	0.930248	0.888321
diode	0.565652	0.551911	0.517006
tbdiv10	0.828887	0.55977	0.669162
tbdiv6	0.850575	0.930248	0.888321
switch	0.828887	0.914068	0.868377
div10	0.63043	0.396804	0.443412
div6	0.668185	0.61758	0.615852
bc10	0.668185	0.710366	0.670076
bc6	0.668185	0.710366	0.670076
secOnesDisplay	0.457441	0.914068	0.713164
secTensDisplay	0.457441	0.914068	0.713164

Average highest gmap weight for low detail dataset

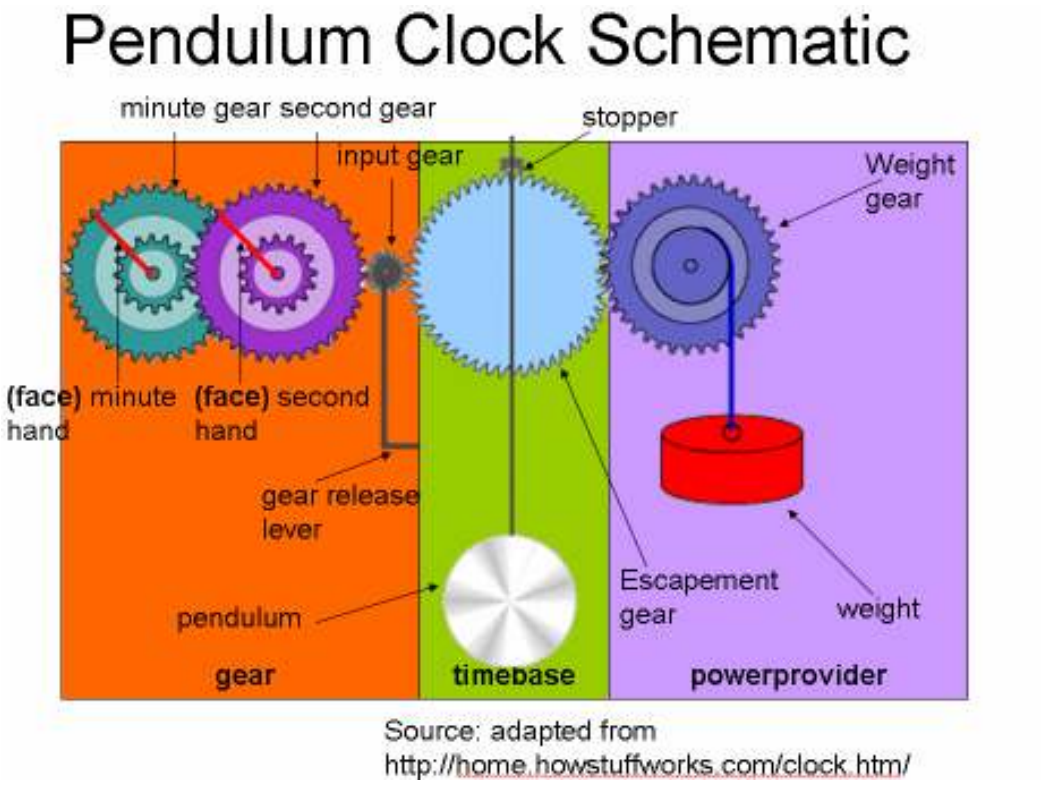
Appendix B Clock Figures For Human Experiment

This appendix shows the schematics for the digital and pendulum clocks that the respondents used during the human experiment. These diagrams are color coded in order to show which clock subcomponents are part of which clock components. For example, in the digital clock schematic, tbdiv10 and tbdiv6 are both in the green box which is marked “timebase.” This shows that the two clock subcomponents, tbdiv10 and tbdiv6, are part of the timebase clock component.

B.1 Digitial Clock Schematic



B.2 Pendulum Clock



Appendix C Questionnaire

This appendix shows the questionnaire used during the human experiment. It also shows the raw data collected during the experiment.

C.1 Questionnaire

Name: _____

Date: _____

Instructions:

- *I'd like you to evaluate the responses that the computer gave when evaluating the same devices you just did.*
- *In particular, these questions are asking you to evaluate the novelty of the matches and the reasons behind the matches.*
- *Each of the 8 questions has output from SME. There are 3 boxes for similar devices, flows, and behaviors.*
- *Mark how novel you think it is on a scale of low, medium, and high*
- *Novelty means how original the match is. If the match is something you would have never thought of yourself, it has high novelty, but if the match is obvious, then it has low novelty.*
- *Do NOT evaluate the correctness of the match. The correctness of the match does not make it any more or less novel.*

1. digital timebase :: pendulum face

___low ___medium ___high

Devices similar:

div10	human
tbdiv10	minutegear
tbdiv6	minutehand

Flows similar:

electric signal	visual signal
electric signal	torque

Behaviors similar:

Transmit signal from tbdiv6 to div10	transmit visual signal from minutehand to human

Transmit signal from tbdiv10 to tbdiv6	transmit torque from minutegear to minute hand

2. digital timebase :: pendulum face

___low ___medium ___high

Devices similar:

div10	Secondhand
tbdiv6	Secondgear

Flows similar:

electric signal	torque

Behaviors similar:

Transmit signal from tbdiv6 to div10	Transmit torque from secondgear to secondhand

3. digital timebase :: pendulum face

___low ___medium ___high

Devices similar:

tbdiv10	minutehand
tbdiv6	secondhand

Flows similar:

electric signal	torque
electric force	torque

Behaviors similar:

import electric force tbdiv10	import torque minutehand
import signal tbdiv6	import torque secondhand

4. pendulumgear :: digitaltimebase

___low ___medium ___high

Devices similar:

gear-release-lever	tbdiv10
secondgear	tbdiv6

Flows similar:

torque	electric signal
force	electric force

Behaviors similar:

import force to gear-release-lever	import electric force tbdiv10
export torque from secondgear	export signal tbdiv6
import torque to secondgear	import signal to tbdiv6

5. pendulumgear :: digitaltimebase

___low ___medium ___high

Devices similar:

minutegear	tbdiv10
secondgear	tbdiv6

Flows similar:

torque	electric signal
torque	electric force

Behaviors similar:

export torque from minutegear	export signal tbdiv10
import torque to minutegear	import electric force tbdiv10
export torque from secondgear	export signal tbdiv6
import signal to secondgear	import electric force tbdiv6

6. pendulum gear :: digital timebase

___low ___medium ___high

Devices similar:

Secondgear	tbdiv6
Minutegear	div10
Inputgear	tbdiv10

Flows similar:

torque	electric signal

Behaviors similar:

transmit torque from secondgear to minutegear	tbdiv6 transmit signal tbdiv6 to div10
transmit torque from inputgear to secondgear	transmit signal from tbdiv10 to tbdiv6

7. pendulum power provider :: digital timebase

___low ___medium ___high

Devices similar:

weightgear	tbdiv6
weight	tbdiv10

Flows similar:

force	electric force
torque	electric signal
force	electric signal

Behaviors similar:

export force from weight	export signal from tbdiv10
import force weight	import electric force tbdiv10
export torque weight gear	export signal tbdiv6
import force weightgear	import signal tbdiv6

8. pendulum power provider :: digital timebase

___low __medium ___high

Devices similar:

escapementgear	div10
weightgear	tbdiv6
weight	tbdiv10

Flows similar:

torque	electric signal
force	electric signal

Behaviors similar:

transmit torque from weightgear to escapementgear	transmit signal from tbdiv6 to div10
transmit force weight to weightgear	transmit signal from tbdiv10 to tbdiv6

C.2 Questionnaire Raw Data

This tables below show the responses for each of the respondents on the questionnaire.

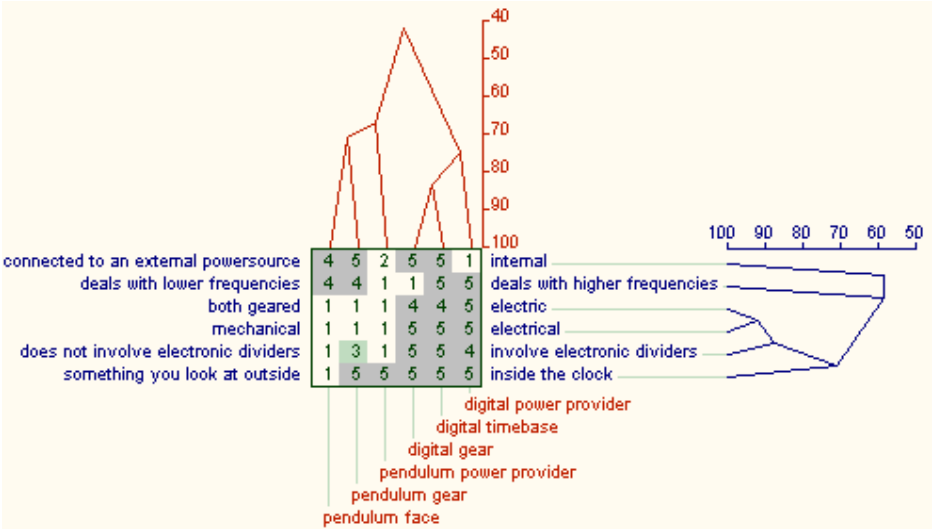
question #	respondent1	respondent2	respondent3	respondent4	respondent5
1	medium	medium	medium	medium	high
2	low	high	low	low	low
3	medium	medium	low	medium	medium
4	medium	high	high	medium	medium
5	medium	low	medium	low	low
6	low	high	medium	high	low
7	low	low	high	medium	high
8	low	medium	medium	high	medium

question #	respondent6	respondent7	respondent8	respondent9	respondent10
1	medium	medium	low	medium	medium
2	low	low	low	low	high
3	medium	high	low	high	low
4	high	high	low	medium	high
5	low	medium	low	medium	low
6	low	medium	low	medium	medium
7	low	high	low	high	high
8	high	medium	low	high	high

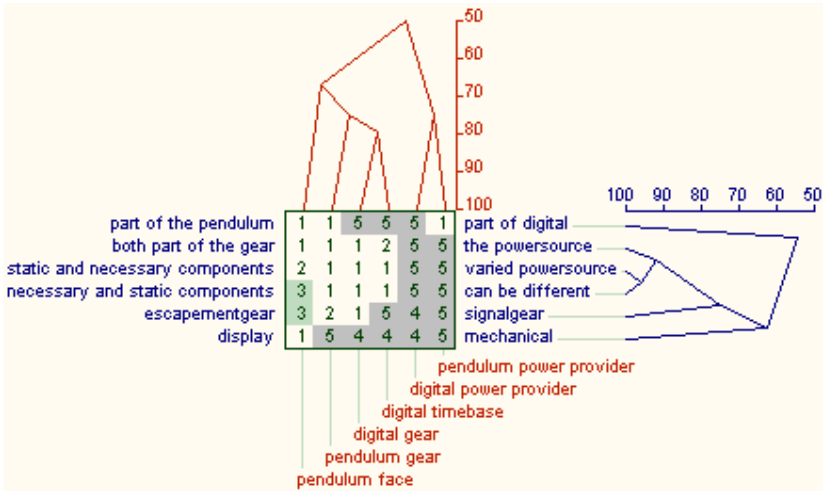
Appendix D Repertory Grid

Appendix D.1 Repertory Grid Data

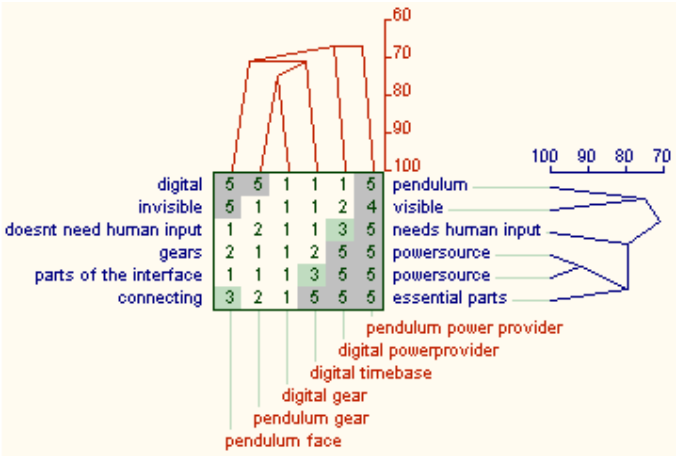
This appendix shows the graphical representation for all the repertory grids collected during the human experiment. Each figure corresponds to a particular respondent number.



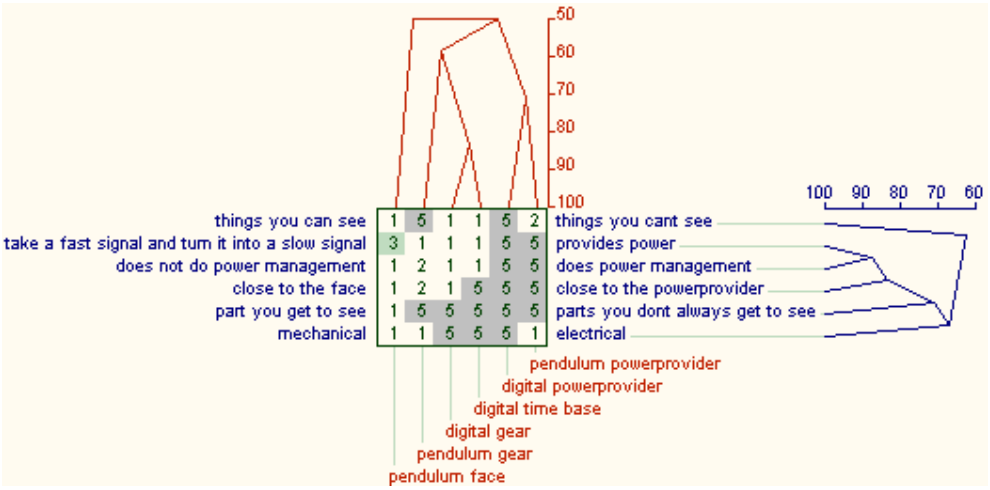
Repertory grid for respondent #1



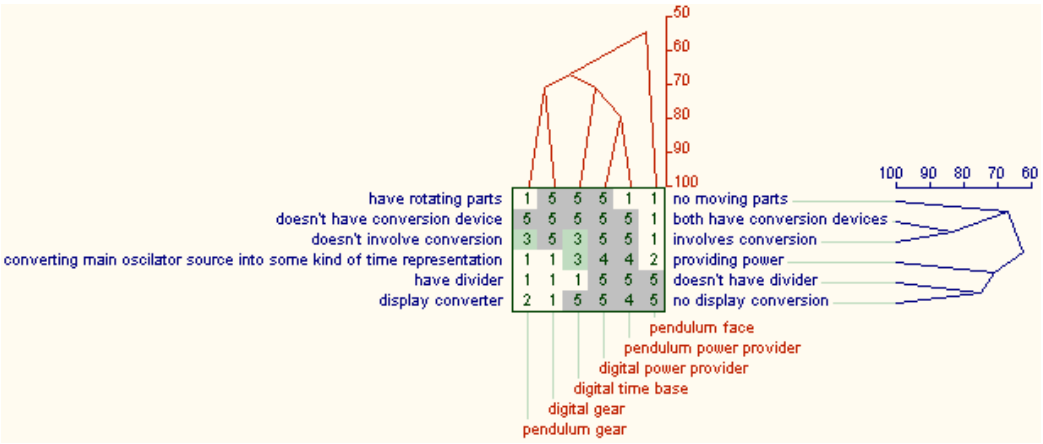
Repertory grid for respondent #2



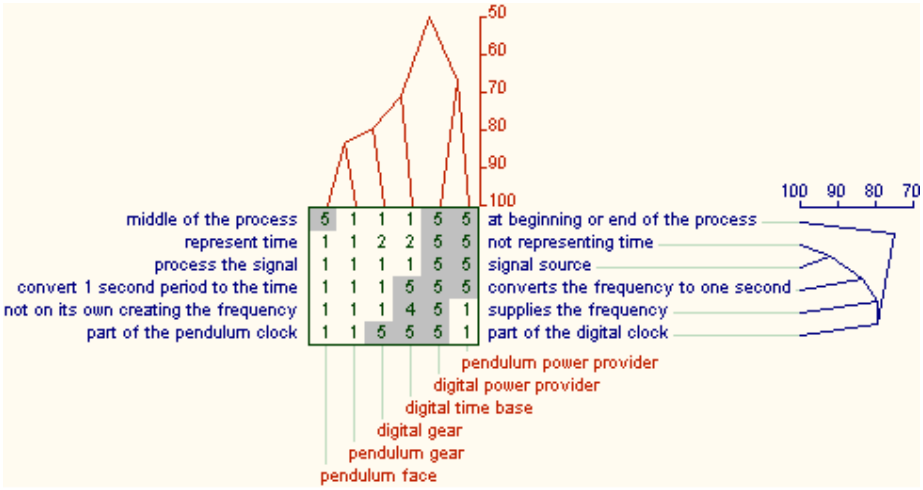
Repertory grid for respondent #3



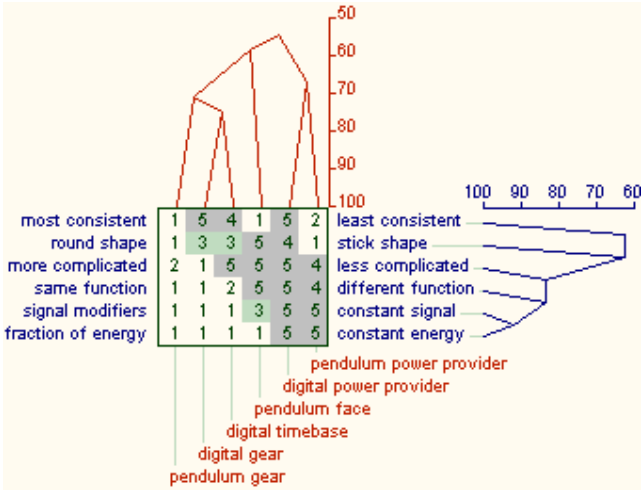
Repertory grid for respondent #4



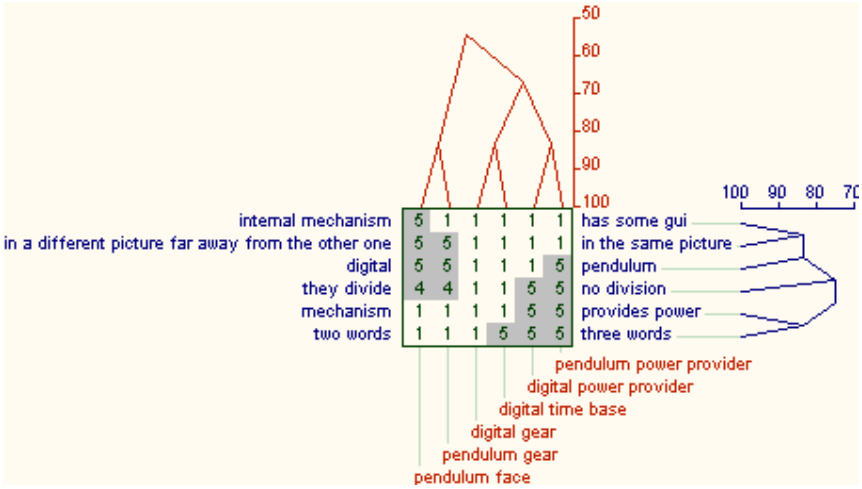
Repertory grid for respondent #5



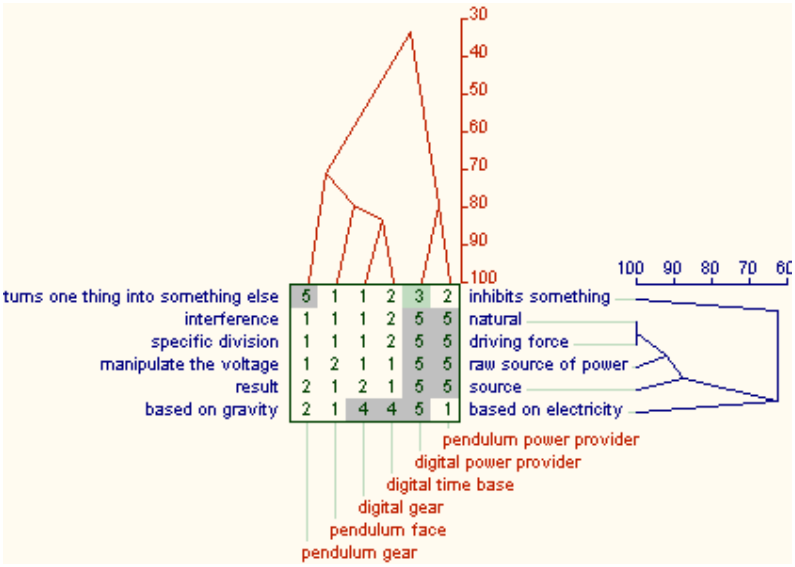
Repertory grid for respondent #6



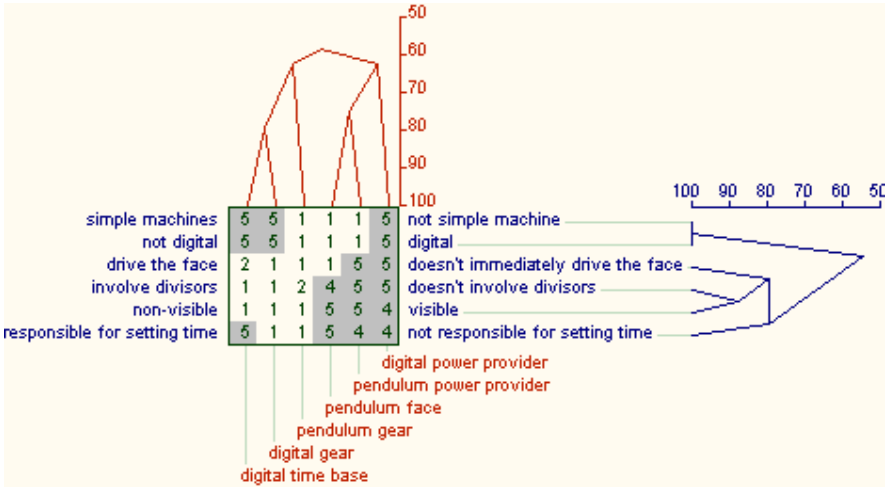
Repertory grid for respondent #7



Repertory grid for respondent #8



Repertory grid for respondent #9



Repertory grid for respondent #10

Appendix D.2 Repertory Grid Construct Categories

This appendix includes all the respondent constructs grouped into themes and then characterized as DC, EC, or neither. The constructs are in table format, where the cells correspond to the construct's respondent #, left and right poles, and DC, EC, or neither characterization.

DC categories

theme: appearance

description: refers to how the device looks.

constructs:

7	round shape	stick shape	appearance	DC

theme: complexity

description: refers to the simplicity, complexity, or consistency of the device.

7	most consistent	least consistent	complexity	DC
7	more complicated	less complicated	complexity	DC
10	simple machines	not simple machine	complexity	DC

theme: has feature

description: describes a feature that the device has like "rotating parts" or "dividers". It is DC because it is referring to something about the device and not referring to how the device is interacting with the environment.

1	does not involve electronic dividers	involve electronic dividers	has feature	DC
5	have rotating parts	no moving parts	has feature	DC
5	doesn't have conversion device	both have conversion device	has feature	DC
5	have divider	doesn't have divider	has feature	DC
10	involve divisors	doesn't involve divisors	has feature	DC

EC categories

theme: conditions of environment

description: refers to something in the environment that is required like human input or gravity.

1	deals with lower frequencies	deals with higher frequencies	conditions of environment	EC
3	doesn't need human input	needs human input	conditions of environment	EC
7	fraction of energy	constant energy	conditions of environment	EC

9	based on gravity	based on electricity	conditions of environment	EC

theme: function of clock

description: refers to how the device relates to the functioning of the overall clock.

6	represent time	not representing time	function of clock	EC
10	responsible for setting time	not responsible for setting time	function of clock	EC

theme: internal versus external

description: refers to how the device is embedded in the clock. It has an implied description of the environment it is in.

1	connected to an external powersource	internal	in/ex	EC
8	internal mechanism	has some gui	in/ex	EC

theme: sequence

description: refers to the device being part of a process.

6	middle of the process	at beginning or end of the process	sequence	EC
9	result	source	sequence	EC

theme: structural significance

description: refers to how the device is positioned within the clock.

2	both part of the gear	the powersource	structural significance	EC
4	close to the face	close to the power provider	structural significance	EC

theme: used in other applications

description: refers to how the device may be used in other environments.

2	necessary and static components	can be different	used in other applications	EC

theme: visible

description: refers to whether or not the device is visible. Since a device can only be visible if it is in an environment, these constructs are marked as EC.

1	something you look at outside	inside the clock	visible	EC
3	invisible	visible	visible	EC
4	things you can see	things you can't see	visible	EC
4	parts you get to see	parts your don't always get to see	visible	EC
10	non-visible	visible	visible	EC

Mixed categories

theme: flow change

description: refers to how an input flow changes into another flow. The construct is DC if the flow change is about input and output. It is EC if the flow change is related to how the resulting flow will be used.

4	take a fast signal and turn it into a slow signal	provides power	flow change	DC
5	converting main oscillator source into some kind of time representation	providing power	flow change	EC
6	convert 1 second period to the time	converts the frequency to one second	flow change	EC

theme: named function

description: a one or two word way of naming the function. Some constructs were EC if they referred to how the device functioned with the other devices in the clock. Some were DC if they only referred to an aspect of the device and not to any role or external thing. For example "connecting vs. essential parts" is EC but "display vs. mechanical" is DC. Also, this category includes one construct that is neither because it is very generic.

2	static and necessary components	varied powersource	named function	EC
2	escapement gear	signal gear(really weightgear)	named function	DC
2	display	mechanical	named function	DC
3	gears	powersource	named function	DC
3	parts of the interface	powersource	named function	EC
3	connecting	essential parts	named function	EC
4	does not do power management	does power management	named function	EC
7	same function	different function	named function	neither

theme: unnamed flow

description: refers to a function, but not what kind of flow it operates on. The construct is DC if it just describes the process that the device is doing like "involves conversion". It is EC if the construct is about how the device is affecting the overall function of the clock like "driving force."

9	turns one thing into something else	inhibits something	unnamed flow	DC
5	doesn't involve conversion	involves conversion	unnamed flow	DC
8	they divide	no division	unnamed flow	DC
8	mechanism	provides power	unnamed flow	EC
9	interference	natural	unnamed flow	EC
9	specific division	driving force	unnamed flow	EC

theme: what it does to a flow

description: refers to performing an operation on a flow. The construct is EC if it refers to something external or more than one device. It is DC if it does not mention how it effects the environment.

5	display converter	no display conversion	what it does to a flow	EC
6	process the signal	signal source	what it does to a flow	EC
6	not on its own creating the frequency	supplies the frequency	what it does to a flow	EC
7	signal modifiers	constant signal	what it does to a flow	DC
9	manipulate the voltage	raw source of power	what it does to a flow	DC
10	drive the face	doesn't immediately drive the face	what it does to a flow	EC

Neither categories

theme: not about the clocks.

description: refers to the way the information about clocks was presented.

8	in a different picture far away from the other one	in the same picture	unclassified	neither
8	two words	three words	unclassified	neither

theme: pendulum vs. digital

description: refers to the difference between being part of the pendulum clock vs. being part of the digital clock.

1	both geared	electric	pendulum vs. digital	neither
1	mechanical	electrical	pendulum vs. digital	neither
2	part of the pendulum	part of digital	pendulum vs. digital	neither
3	digital	pendulum	pendulum vs. digital	neither
4	mechanical	electrical	pendulum vs. digital	neither
6	part of the pendulum clock	part of the digital clock	pendulum vs. digital	neither
8	digital	pendulum	pendulum vs. digital	neither
10	not digital	digital	pendulum vs. digital	neither