Beyond Function-Behavior-Structure

Mahmoud Dinar, Chris Maclellan, Andreea Danielescu, and Jami Shah
Arizona State University, USA

Our research is investigating the relationship between design problem formulation and creative outcome. Towards that goal we have conducted experiments with designers engaged in problem formulation. In order to analyze such empirical data, a formal representation is needed. One popular model is the Function-Behavior-Structure (FBS) model and its several variants. Our problem map (P-map) model shares many common features with FBS but also many differences. A symmetric hierarchical representation is introduced not only in each of the three domains (F, B, S) but in additional domains (requirements and issues). Generic inter and intra-domain relationships between these entities are also identified in addition to optional attributes of the entities, leading to a more expressive and flexible model that is domain independent and well suited for representing problem formulations of designers with different expertise levels and creativity. The model has been used for coding protocol data in a formal predicate logic language (Answer Set Programming).

Introduction

The main objective of our research is to investigate how design problem formulation is related to creativity. We have conducted exploratory protocol studies with designers engaged in problem formulation. In order to analyze such empirical data and utilize further investigations, a formal representation and a more efficient data collection method are needed.

We began by creating a model based on the observations from an exploratory protocol analysis [1]. We identified some entities in problem formulation such as function, component, analogy, question etc. Instances of each entity appeared in order of emergence with unique tag names and

Design Computing and Cognition DCC'12. J.S. Gero (ed), pp. xx-yy.
© Springer 2012

wherever there was a relation between two instances of two different entities, a line was drawn to show the link. The first model suffered from an incoherent grouping of entities and specific relation types. In the second version, we modified the model to capture more specific relationships. We have called our model the *Problem map (P-map)* [2].

We were concerned mainly with two aspects of the framework we built our model in: the ontology, i.e. the entities, their attributes, and the relationships among them; and the formalism for representing the data. In defining the entities we looked at other well-known models in design research. One popular model is the Function-Behavior-Structure (FBS) model [3] and its several similar variants such as the SBF model [4] and the Functional Representation model [5].

In an ongoing effort to capture pathways of problem formulation towards creative design, we have changed both the ontology and the notation in our *P-map* from the previous version. Even though our model has shared many common features with FBS, it has many differences. These differences arise from potential ways of improving FBS to be incorporated in new avenues in design research, especially pre-ideation stages, i.e. problem formulation. We focus on improving flexibility, representation of hierarchical structure, and expressiveness of the model. We use Answer Set Programming (ASP) [6], a formal predicate logic language, for representation.

Function-Behavior-Structure models

FBS is an established framework in design research that has prompted a great body of work within the past two decades since its introduction. The concepts of the model, namely the main three entities and the relations among them, were not unfamiliar themes in engineering design. Around the same time Gero [3] published his seminal paper, Gui [7] for example proposed a scheme for representing mechanical components and assemblies that used predicate logic to describe function, behavior and structure in an object model.

The extensive work related to FBS in engineering design research deserves a dedicated survey to the subject. However, a few useful critiques of the model help to that end. Dorst and Vermaas [8] take a dialectical stance and follow the changes in the definitions in the FBS model in three papers of Gero's work. They search for distinctions between intentional descriptions of artifacts and structural descriptions, and conclude that the changes proposed to improve or clarify the definitions were actually prompted by changes in the objectives that the model was supposed to serve. They argue that the shift from a descriptive model to a prescriptive model was a source of ambiguity. Gero did not seem to claim that his model laid focus on either of the directions in his research. If the model can be used to both ends, that should be considered an advantage of it. In our research we have tried to create a flexible model of design, however, we too do not claim that our model can be used in all phases of the design process, for all types of design problems, or solely for descriptive or prescriptive models.

Galle [9] finds the modified definitions that Dorst and Vermaas propose incomplete because they refer to an artifact that does not yet exist. Although he supports the separation of intentional and non-intentional concepts and the addition of *purpose* in their modified FBS, he offers two distinct definitions for FBS. On the one hand he defines a *nominalist* model that is purely logical and symbol-oriented. On the other hand the *realist* model sees design and thus FBS as a mathematical abstraction of experiential knowledge about the behaviors that embody structure in terms of material objects.

Besides FBS, other variants of the model have been developed independently. The most noteworthy of these variants are the Functional Representation (FR) and SBF models. Chandrasekaran [5] developed FR as a language which described the function of an artifact in terms of causal processes in order to simulate, diagnose or explain how the artifact works. In his later work with Josephson [10], he added an environment-centric view in addition to the original device-centric view to allow for more precise ways of representing design knowledge. In describing how FR can benefit from attempts in finding a common functional modeling (FM) ontology [11, 12], he argues that artificial intelligence research that aims to model artifacts can lend FM becoming a more formalized representation [13].

Goel et al. [4] have developed the SBF modeling language for a teleological description of complex systems. In this language, structure, behavior and function are represented in terms of components and their connections, transitions among a sequence of states, and pre- and post-conditions respectively. The syntax is similar to notations that are used to represent production rules. The model is a top-down description scheme, in which each fragment of the model is defined by a lower level fragment. At the top there is an instance of SBF while at the bottom there are the building block fragments such as strings and integers. For example, an *element* (or in other words a component in a structure model) is defined by an integer *Id*, a string *name*, a string *description*, an optional set (can have zero number of fragments) of *property*, and an integer *subelement Id*.

FBS has mainly been used in modeling the design process, protocol analysis, and design automation. Gero [3] identified activities in the design process in terms of transformations from one of the three domains to another, considering a difference between expected and actual behavior. For example, transforming a function to an expected behavior is considered formulation or specification. Gero and Kannengieser [14] took into account the dynamic character of design by considering the notion of situatedness. They extended FBS further to the explicit modeling of design processes [15]. Howard et al. [16] adopted FBS to describe a creative design process proposed by integrating engineering design process views and creativity from the standpoint of cognitive psychology.

Just as FBS has been used as an attempt to create a common consensus among design researchers in defining fundamental concepts, it has also been used as a coding scheme in analyzing protocol studies [17–19]. The application of such ontological description has been extensively used in developing design automation. Umeda et al. [20] proposed a computer tool Function-Behavior-State Modeler for supporting synthesis in the conceptual design stage. Anthony et al. [21] integrated engineering knowledge as a description of function and behavior with a semantically annotated representation of 3D graphical models. For a list of some design automation systems that have used SBF see Goel et al. [4]. Maher and Gomez de Silva Garza [22] also provide examples of application of FBS in case-based reasoning tools in design.

Aspects to consider in improving the model

Inspired by the FBS framework, we realized that we should consider a few aspects to change in order to capture the problem formulation space more efficiently. We explain why FBS needed flexibility, hierarchy and expressiveness.

Flexibility

Dorst and Vermaas [8] argue that the changes in the definition of the entities in FBS were made mostly to accommodate a distinction between an intentional and a structural purpose of a design. One might argue that adhering to such distinction may not offer significant advantages in defining reasoning mechanisms. Moreover, design activities are often separated from manufacturing. It is convenient to assume design ends with a document that specifies design, as Gero assumed in [3]. However, regardless of the difference in how intended or actual behavior achieves a function, not all of what is expected from a design can be solely defined in terms of its functions. Consider the example of designing an artifact that moves a load from one point to another: if one ignores the conditions of the environment, load capacity and distance should still be specified. Such requirements are obviously, not functional requirements.

Umeda et al. [23] stated that function, structure and behavior represent a high to low level of abstraction, respectively, that is, the emergence of behavioral descriptions follow that of the structure and function monotonically and away from abstract concepts. However, this is not always the case in design practice. For example, a behavior is not always expressed in terms of rigid mathematical equations. Sometimes the designers' knowledge resembles qualitative physical expressions. For example the designer may only be aware of an existing relationship among different parameters without knowing the specifics of that relation. A more flexible model allows the abstraction of each of these entities.

Additionally, in real life experience, design data fragments do not appear monotonically. If there were attributes that defined the entities in FBS models, rarely are these attributes completely filled once a new instance of an entity is added. In the SBF modeling language [4] there is an extensive description of each entity and some of the attributes are optional sets. However it seems the addition of new attributes in an instance of an entity alters the initial object entirely.

Hierarchy

The ability to flexibly represent a hierarchical structure is instrumental to modeling complex and evolutionary systems [24]. Products are getting ever more complex and design is inherently an evolutionary process [25, 26]. In addition, hierarchies often help define levels of abstraction. Design studies also agree that designers move among different levels of abstraction intermittently during design [27, 28]. As explained before, the need for flexible representation of different levels of abstraction is necessary in capturing problem formulation data. Abstraction is not limited to different entities but also in a hierarchy of any of the entities. The hierarchical structure spans all the main entities. Requirement elicitation is as common an approach in design as functional decomposition is. Artifacts especially in complex and adaptable designs have a hierarchical structure, where sometimes components are independent modules. Therefore a model in the FBS framework that is restrained to function, behavior, and structure descriptions without representing compositions and multiple levels of abstraction at each entity, is not sufficient.

Expressiveness

Gero and Kannengieser [29] state that FBS ontology is a high level model. FBS still can benefit from a more elaborate definition of attributes to enhance its expressiveness. Parametric relations for example can be described in more detail in behavior at different levels of abstraction, i.e. in terms of mathematical expressions, qualitative models, or unknown relations as explained before.

As stated earlier about there is a need for a more flexible model to explicitly expressing non-functional requirements. Other details such as the importance of the requirements (e.g. in terms which are often used such as musts and wishes), their sources (e.g. customer needs, regulations, conditions of the operating environment), and the mating conditions among components help improve capturing more of the formulation space and thus opening potential ways for the discovery of paths to creative outcome.

Our proposed model

To address the points we made above, a symmetric hierarchical representation is introduced not only in each of the three domains (F, B, S) but in additionally proposed domains, which are requirements and issues. We use the term *artifact* instead of *structure* to reserve the term *hierarchy* as a common characteristic of all of the main domains in the model, and to emphasize domain independence.

Although modeling design processes is not our objective, it is convenient to explain our model following a typical design process. All new designs start with explicitly specified requirements, which include what the artifact is supposed to do (its main or highest level function); the performance levels desired (technical specifications) and overarching design goals. To that, the designer applies his domain knowledge which includes procedural knowledge (functional decomposition, search strategy, etc.) and artifact knowledge (candidate solutions, physical laws governing behavior). From an analysis of the requirements the designer may gain key insights, particularly in the discovery of problematic issues. This is consistent with our earlier observations [1, 2].

From these observations we can say that at the most general level, problem definition elements can be grouped into the following categories: *Requirement*, *Function*, *Artifact*, *Behavior*, and *Issue*. The groups are built around base entities and their hierarchical structure in terms of upper-lower level, and preceding-succeeding entities. The base entities in functions, requirements, and issues share the name of the group. In addition to the hierarchies and the directional (sequential) or non-directional relations among them there are supporting entities such as parametric relations under the group *Behavior*. All groups except the *Issue* group are related with bidirectional relations, while the *Issue* group can have a relation to any combination of the rest of the entities. Before we explain the details of our model any further we should talk about the formalism that we used for our representation. Notice that entity names are italicized in lower case while group names are in upper case.

The formalism

The improved P-map model has been utilized to code protocol data, which necessitates the use of a consistent formalism. The formalism that we chose to encode P-map data was Answer Set Programming (ASP) [6]. ASP, a non-monotonic declarative logical programming language, is theoretically based on Answer Set semantics [30]. Similar to Prolog, Answer Set Programs consist of two main components: facts, which are the ground literals over which the system reasons, and rules, which are used to perform logical reasoning over the facts. An Answer Set Program solver finds all of the answer sets which are entailed by the given set of facts and rules. Similarly, our answer set representation is a simple way to codify protocol data. On the other hand, our answer set representation provides a means to reason about the coded *P-maps*. For this paper, we focus solely on representing protocol data in a concise way. Our primary reasons for choosing Answer Set Programming were the simplicity of the logical formalism, the direct mapping between the *P-map* and an Answer Set representation, and the ease with which we could perform automated reasoning over the represented *P-map*.

The main ingredient of the answer sets are the predicates. A predicate is represented with a tag name followed by braces in which there are the values of the attributes that define the predicate. Since ASP representation is declarative and monotonic, only the mandatory attributes of an entity reside in the predicate that define it. For example, a *solution principle* which is one of the entities in the *Artifact* (S in FBS) group, is represented as:

```
solutionPrinciple(<Id>,<description>)
```

When used, the actual values would be substituted into the *Id* and *description* spots. The IDs (which are unique) start with two letters corresponding to an entity. In an example that shows a design of an airplane seat that can be automatically turned into a bed, a solution principle may look like this:

solutionPrinciple(sl_telescope,"pads inserted into one another come out of the seat")

As another example, consider the *function*, which is the main entity of the *Function* group. To keep the model simpler, the Ids are attributes that may represent the entity in their own right. The Id for a function is taken as its action verb:

function(fn_moving_to_flat_position)

In addition to these main components, there are a number of supporting predicates, some of which are common among several groups. The hierarchical structure is defined by:

parentOf(<parent>,<child>,<hierarchy>)

Sequences are defined by:

before(<preceding>,<succeeding>)

A functional decomposition for the airplane seat example is presented as follows:

```
function(fn_supporting_sleeping)
function(fn_moving_to_flat_position)
function(fn_supporting_in_flat_position)
parentOf(fn_supporting_sleeping, fn_moving_to_flat_position,hy_fn1)
parentOf(fn_supporting_sleeping, fn_supporting_in_flat_position,
hy_fn1)
before(fn_moving_to_flat_position, fn_supporting_in_flat_position,
hy_fn1)
```

The hierarchy Id is used to allow for disjunctive decompositions and make the model more flexible.

The entities

In all the five groups there is a hierarchical structure. In the *function* group, the hierarchy is evident in functional decompositions. Our model incorporates disjunctive composition, making it possible to have multiple functional decompositions using common sub-functions.

In the *artifact* group, the product architecture is a hierarchy of physical embodiments and solution principles. Our model also allows partial compositions of solution principles and physical embodiments, since in reality, the designer follows different parts of the sub-solutions at different times corresponding to different levels of abstraction. Similarly in the *requirement* group, requirements and goals can participate in the same hierarchy.

In the *behavior* group, a physical effect is a hierarchy of physical laws. Physical effects may be expressed by parametric relations, which are composed of sets of parameters. In the *issue* group, the hierarchy entails the priority the designer gives to the issues that should be addressed, that may correspond to their problem solving strategy. The ontology of our *P-map* model can be seen in Figure 1 in terms of Barker notations [31].

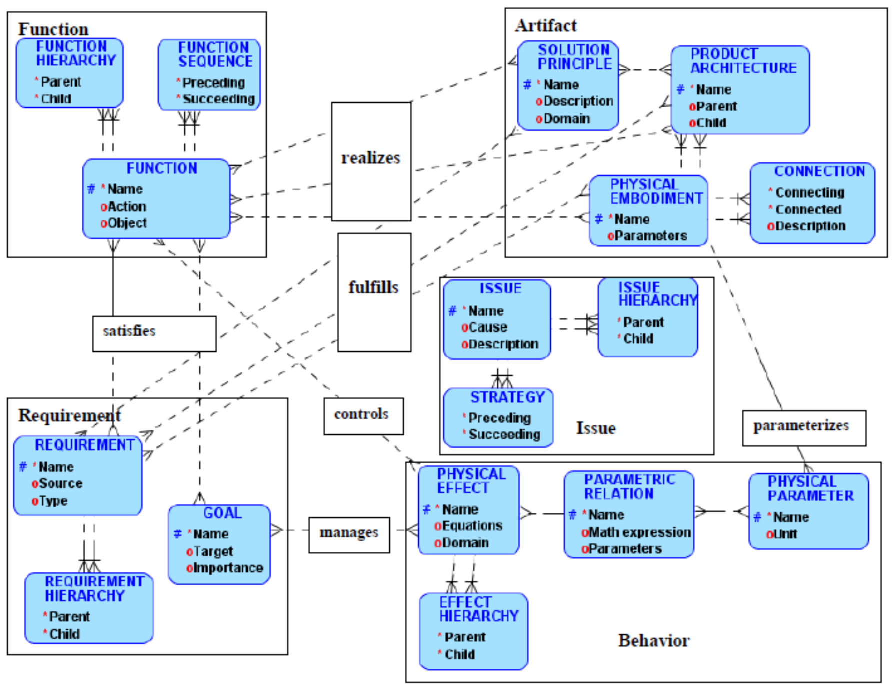

Figure 1. The ontology of the *P-map* model

As it can be seen, the names that are given to the hierarchies are different in some groups to reflect common terminology in design corresponding to the same data structure. In the *Artifact* group, the *product architecture* is the hierarchical structure accommodating a composition of physical embodiments, i.e. components, and solution principles. The *connection* entity represents mating conditions among components and since it is a non-directional sequence, a different predicate from the *before* predicate, defined as connects(<em_Id/sl_Id>,<em_Id/sl_Id,hy_Id>), is used for it.

We explain the additional entities to the FBS model, i.e. *requirement* and *issue*, and the inter-group relations in more detail.

Requirements

Different definitions and interpretations of FBS have tried to place the intention or the purpose of the design along the lines of any of the three domains [8]. Design problems, however, are different in that they do not always start with a known functional decomposition that requires a search for components for each function, or with known components that should be adapted to accommodate new functions. Requirements, especially in the design of novel products for specific customer needs, should be explicitly identified.

A design problem is usually given as a design brief or problem statement. The design problem, however, is defined with additional requirements elicited by the designer. Therefore, the problem is specified by a set of design goals and requirements. A desired design is one that realizes all the requirements and achieves better design goals. In our model, the set of design requirements and goals is defined as follows:

```
requirement(<Id>)
requirementSource(<rq_ Id> ,< source>)

goal(<Id>)
goalImportance(<gl_ Id> ,< importance>)
goalTarget(<gl_Id>,<target>,<improvement_direction>)
```

The source can be defined by specific tag names that might be set for the user to choose from: "safety", "aesthetics", "ergonomics", "use_environment", "affordance" etc. The source might also specify whether the requirement was explicitly given in the problem statement or it was discovered by the designer.

We assume that requirements are binary; they are either met or not met. Goals on the contrary are defined with their importance level and the targets with ranges of values that should be achieved. They might have a single bound or be discrete. There is usually a relation between the level of satisfaction in using the solution and the degree to which the goal is achieved in terms of a utility function. The improvement direction can take one of the three values: "more", -the goal is desired to be more than the given target; "less", - the goal is desired to be less than the given target; "about", - the goal is desired to be as close to the given target as possible.

Issues

An issue is a point that the designer believes to be pivotal or problematic in achieving a design objective. An issue can arise in realizing a function with a specific artifact or behavior, in realizing conflicting design goals such as lower weight and strength of a structure or in accommodating different components in a product architecture due to incompatible interfaces to name a few. Issues, therefore, can be seen as intimately related to the other domains. The designer gains insight in the discovery of key issues in the design and the areas of the design that should be prioritized. The hierarchy of issues may represent problem solving strategy. To avoid repeating the same example, we show this hierarchal structure in the next section when we talk about the relations. A list of entities and their attributes in addition to corresponding entities from the FBS framework are given in Table 1.

Table 1. A summary of entities and their attributes

Group	**Entities**	**Attributes**	**Corresponding to FBS**
Requirement	Requirement	Id, importance, source, domain	Purpose/ Function
	Goal	Id, importance, target, direction	
Function	Function	Id, parameter	Function
Artifact	Solution principle	Id, description, domain	Structure
	Physical embodiment	Id, parameter	
	Parameter	name, unit, value	
Behavior	Behavior	Id, domain	Behavior
	Parametric relation	Id, equation, abstraction level, parameter	
Issue	Issue	Id, description	-----------------

Distinct relations

Generic inter and intra-domain relationships between these entities are also identified in addition to optional attributes of the entities. This leads to a more expressive and flexible model that is domain independent and well suited for representing problem formulations of designers with different expertise levels and creativity. Table 2 shows the inter-group relations among entities.

Table 2. Inter-group relations

Entity	Relation	Entity
Function	satisfies	Requirement
Artifact	fulfills	Requirement
Behavior	manages	Requirement
Artifact	realizes	Function
Artifact	parameterizes	Behavior
Behavior	controls	Function
Issue	relates	All entities and their combinations

Issues can be related to any combination of other entities. For the airplane seat example, design issues in our model can be represented as the Id for an issue starts with iu and the Id for a physical behavior starts with ph):

issue(iu_comfortable_support_at_flat_position,"deflection")
issue(iu_covering_length_at_flat_position,"increasing number of boxes")
issue(iu_support_weight_at_flat_position, "load on a cantilever causes high bending stress")

relates(iu_comfortable_support_at_flat_position, fn_supporting_in_flat_position)

relates(iu_covering_length_at_flat_position, rq_length_of_seat_cushion_less_than_20in)

relates(iu_covering_length_at_flat_position, sl_telescope)

relates(iu_support_weight_at_flat_position, ph_bending_stress)

relates (iu_support_weight_at_flat_position, rq_support_250lb_weight)

relates (iu_support_weight_at_flat_position, sl_pivoting_recliner)

parentOf(iu_comfortable_support_at_flat_position,iu_covering_length_at_flat_position, iu_hy1)

paren-
tOf(iu_comfortable_support_at_flat_position,iu_support_weight_at_flat_p
osition, iu_hy1)

before(iu_support_weight_at_flat_position,
iu_covering_length_at_flat_position, iu_hy1)

Discussion

The main objective of our research is to discover the relation between problem formulation and creative design. Design has common cognitive characteristics to human problem solving [32]. Newell and Simon [33] described a computational framework for studying cognitive aspects of human problem solving in terms of states, operations and goals. Though it is plausible to create models to investigate all cognitive aspects of design, it might be more efficient to focus on parts of them. We focus on modeling problem formulation states and we believe that our *P-map* model can be useful for such purpose.

In real life, design is not monotonic or procedural. The designer at each step of the design may think about new solution principles, using alternate behaviors and corresponding functions, or add new requirements. A benefit of implementing a declarative computational framework for our representation is that partial data fragments can be added and deleted without worrying about consistency.

What can the *P-map* be used for

In line with our research objectives, we intend to improve our method of empirical investigation. Additionally, we are searching for pathways that lead towards creative design by comparing problem formulations of designers of different expertise and creativity levels.

We are building an interactive tool for collecting data about solving design problems, in our *P-map* format. The tool is not the focus of this paper.

In our earlier work, we have tried to find a representation that captures differences among how designers formulate problems in terms of changes in data fragments over time, i.e. formulation states. At this point, we are not making any hypothesis about the relation between creativity and formulation. Our *P-map* model, however, can help achieve this goal by providing a fine level analysis tool. Some possible examples, inspired by our earlier discoveries [2] may point to finding patterns in the sequence of the way solution principles and parameters emerge. Another example is the coverage of issues; in other words, one might argue that the less entities related to an issue, the less justified it is, leading to less creative design or an unnecessary inhibition.

When encoding P-maps in an Answer Set formalism, all of the various entities and connections between them are represented as facts. A given Answer Set representation of a *P-map* by itself will contain no rules. This means that when querying the set representation of a *P-map* with a solver, only one answer set is returned, the answer set containing the facts that we gave our system. In future work, rules could be developed which govern how the system should reason over the *P-map*. This will result in more interesting behavior when executing the answer set program. An example is the number of different solution principles that have mathematically expressed behaviors that satisfy all the requirements. Another example is the unresolved issues that may in essence be inhibitory. This means that the designer considers an issue in a narrow view e.g. in satisfying a requirement by a solution principle without understanding that there is a related behavior that should be examined as well.

How can the *P-map* be examined and validated

We propose a data model to capture problem formulation in design in the form of a sequence of state representations. There are two aspects for a data model, formalism and content.

Developing formalism imposes requirements on a data model. We describe some of the requirements that we set for our data model.

Domain independence

The model should not be limited to representing a specific class of objects. For example, it should accommodate the design of a combustion engine with its known behaviors, or an engine with a general function of providing power, including but not limited to a solar-powered engine. We focus on mechanical devices in our examples but they may have electronics and software.

Compactness

The model should have a simple and compact representation. This is a relative measure but it provides hints for including some entities with similar properties in the same group or class. For example, we consider safety and ergonomics as requirements. We need to incorporate our model in a software tool, or translate it to/from a textual configuration such as ASP, hence compactness.

Richness

We want to show creative vs. non-creative designer problem formulation and we assume we can capture that in comparing their P-maps. Thus our data model should not only be compact enough to be easily created or translated, but also it needs to be rich enough to provide such contrasts.

Unambiguity

We need a formalism that can be communicated among interpreters who will code problem formulation data in design. Though there may be differences in labeling the data, a design episode should lead to close enough P-maps for different coders. Thus the entities should be semantically distinct. This also gives the possibility of automatic coding, and interactive conversation for our tentative aid software tool.

Flexibility

Our model should accommodate incomplete or redundant data structures, reflecting inferior design practice. However, the tentative aid software tool should be able to detect such patterns. For example two functional hierarchies with a common parent and common children may coexist (F1 is a parent of f2, f3; F1 is a parent of f2, f3, f4). In addition, the model should be able to hold relations among entities, representing different levels of abstraction. For example a function can be related to a physical embodiment or a product architecture including that physical embodiment.

We refer to entities similarly as we do to classes in Object-Oriented data models. However, we use the notion of relation tables in Relational Database models. For example we want to capture different instances of function sequences or product architecture. We represent such instances with an entity which is represented as a composition or aggregation in an OO model and as a relation table in an RD model.

We may further develop relations among the entities we selected. However, with the compactness criterion we avoid showing such relations as new entities.

We then assign attributes to these entities. Some of the entities in our older models can now be defined as attributes which helps achieve compactness. For example a physical rule (either in an abstract form or as a mathematical equation) may be considered an attribute of a behavior.

To validate these data model requirements, we will measure the expressiveness of the model by examining how much of protocol data can be expressed in terms of the entities and their relations. Additionally, we will measure inter-rater agreement; we check how different two raters analyze a given protocol.

Conclusion

In order to investigate the relationship between design problem formulation and creativity, we have created a data model called the *P-map*. In defining its entities we looked at other models in design research. We were inspired by the Function-Behavior-Structure (FBS) model and its several variants such as the SBF model and the Functional Representation model. The *P-map* shares many common features with FBS, although it has many differences. Similarities include the common function, behavior and structure (artifact) group of entities and the way behaviors link functions and structures. The differences are: allowing disjunctive hierarchical compositions; the explicit declaration of requirements; the consideration of issues designers face in meeting requirements, achieving functions, controlling behaviors, or in any combination of these states; and the distinct relations among the groups. These differences between our *P-map* and FBS are due to potential ways of improving FBS to be incorporated in problem formulation. We have focused on improving flexibility, representation of hierarchical structure, and expressiveness of the model.

The ability to flexibly represent a hierarchical structure is instrumental to modeling complex and evolutionary systems. Products are getting ever more complex and design is inherently an evolutionary process. In addition, hierarchies often help define levels of abstraction. Design studies also agree that designers move among different levels of abstraction intermittently during design.

With a flexible model that represents hierarchical structure, FBS can still benefit from a more elaborate definition of attributes to enhance its expressiveness. Parametric relations for example can be described in more detail in behavior at different levels of abstraction, i.e. in terms of mathematical expressions, qualitative models, or unknown relations.

To address these points, a symmetric hierarchical representation is introduced not only in each of the three domains (F, B, S) but in additional domains, which are requirements and issues.

Different definitions and interpretations of FBS have tried to place the intention or the purpose of the design along the lines of any of the three domains. Design problems, however, are different in that they do not always start with a known functional decomposition that requires a search for components for each function, or with known components that should be adapted to accommodate new functions. Requirements, especially in the design of novel products for specific customer needs, should be explicitly identified.

An issue is a point that the designer believes to be pivotal or problematic in achieving a design objective. An issue can arise in realizing a function with a specific artifact or behavior, in realizing conflicting design goals such as lower weight and strength of a structure or in accommodating different components in a product architecture due to incompatible interfaces to name a few. Issues, therefore, can be seen as intimately related to the other domains.

Generic inter and intra-domain relationships between these entities are also identified in addition to optional attributes of the entities. This leads to a more expressive and flexible model that is domain independent and well suited for representing problem formulations of designers with different expertise levels and creativity.

The model has been used for coding protocol data which necessitates the use of formalism. Answer Set Programming (ASP) as a formal predicate logic language was used for coding data.

In real life, design is neither monotonic nor procedural. The designer at each step of the design may think about new solution principles, using alternate behaviors and corresponding functions, or add new requirements. A benefit of implementing a declarative computational framework for our representation is that partial data fragments can be added and deleted without worrying about the consistency of the P-map.

We are building an interactive tool for collecting data about solving design problems, in our P-map format. We believe that our *P-map* model can help testing hypotheses about the relation between formulation and creativity by providing an analysis tool at a fine level of detail. Some possible examples, inspired by our earlier discoveries, are finding the sequence of emergence of solution principles, and checking how issues are covered in relation to other entities.

Acknowledgments

This study is supported by the National Science Foundation, CMMI grant number 1002910. The opinions expressed in this paper are those of the authors and are not endorsed by the National Science Foundation. We thank Glen Hunt for his comments on the earlier versions of our *P-map*.

References

1. Dinar M, Shah JJ, Langley P, Hunt GR, Campana E (2011) A Structure for Representing Problem Formulation in Design. Proceedings of the International Conference on Engineering Design

2. Dinar M, Shah JJ, Langley P, Campana E, Hunt GR (2011) Towards a Formal Representation Model of Problem Formulation in Design. Proceedings of ASME DETC

3. Gero JS (1990) Design prototypes: a knowledge representation schema for design. AI Magazine 11(4):26-36

4. Goel AK, Rugaber S, Vattam S (2009) Structure , Behavior and Function of Complex Systems : The Structure, Behavior, and Function Modeling Language. Artificial Intelligence for Engineering Design, Analysis and Manufacturing 23(1):23-35

5. Chandrasekaran B (1994) Functional Representation: A Brief Historical Perspective. Applied Artificial Intelligence 8(2):173-197

6. Michael G (2008) Answer Sets. In: Frank van Harmelen VL and BPBT-F of AI (ed) Handbook of Knowledge Representation. Elsevier, pp 285-316

7. Gui JK (1990) A Function-Behaviour-Structure Machine Design Model and its use in Assembly Sequence Planning. Journal of Engineering Design 1(3):239-259

8. Dorst K, Vermaas PE (2005) John Gero's Function-Behaviour-Structure model of designing: a critical analysis. Research in Engineering Design 16(1-2):17-26

9. Galle P (2009) The ontology of Gero's FBS model of designing. Design Studies 30(4):321-339

10. Chandrasekaran B, Josephson JR (2000) Function in Device Representation. Engineering with Computers 16(3-4):162-177

11. Stone RB, Wood KL (2000) Development of a functional basis for design. Journal of Mechanical Design 122(4):359-370

12. Hirtz J, Stone RB, Mcadams DA, Szykman S, Wood KL (2002) A functional basis for engineering design□ : Reconciling and evolving previous efforts. Research in Engineering Design 13:65-82

13. Chandrasekaran B (2005) Representing function□ : Relating functional representation and functional modeling research streams. Artificial Intelligence for Engineering Design, Analysis and Manufacturing 19:65-74

14. Gero JS, Kannengiesser U (2004) The situated function-behaviour-structure framework. Design Studies 25(4):373-391

15. Gero JS, Kannengiesser U (2007) A function–behavior–structure ontology of processes. Artificial Intelligence for Engineering Design, Analysis and Manufacturing 21(04):379-391

16. Howard TJJ, Culley SJJ, Dekoninck E (2008) Describing the creative design process by the integration of engineering design and cognitive psychology literature. Design Studies 29(2):160-180

17. Gero JS, Mc Neill T (1998) An approach to the analysis of design protocols. Design studies 19(1):21–61

18. Neill TM, Gero JS, Warren J (1998) Understanding conceptual electronic design using protocol analysis. Research in Engineering Design 10(3):129-140

19. Pourmohamadi M, Gero JS (2011) LINKOgrapher: An Analysis Tool to Study Design Protocols Based on FBS Coding. Proceedings of the International Conference on Engineering Design. Copenhagen, Denmark, pp 1-10

20. Umeda Y, Ishii M, Yoshioka M, Shimomura Y, Tomiyama T (1996) Supporting conceptual design based on the function-behavior-state modeler. Artificial Intelligence for Engineering Design, Analysis and Manufacturing 10(4):275-288

21. Anthony L, Regli WC, John JE, Lombeyda SV (2001) An Approach to Capturing Structure, Behavior, and Function of Artifacts in Computer-Aided Design. Journal of Computing and Information Science in Engineering 1(2):186-192

22. Maher ML, Gomez de Silva Garza A (1997) Case-based reasoning in design. IEEE Expert 12(2):34-41

23. Umeda Y, Takeda H, Tomiyama T, Yoshikawa H (1990) Function, behaviour, and structure. Applications of artificial intelligence in engineering V. Computational Mechanics Publications and Springer-Verlag, Berlin, pp 177–194

24. Simon HA (1996) The Sciences of the Artificial. MIT Press, Cambridge, MA

25. Dorst K, Cross N (2001) Creativity in the design process: co-evolution of problem–solution. Design Studies 22:425-437

26. Maher M, Poon J, Boulanger S (1996) Formalising Design Exploration as Co-Evolution: A Combined Gene Approach. Advances in Formal Design Methods for CAD: Proceedings of the IFIP WG5.2 Workshop on Formal Design Methods for Computer-Aided Design

27. Eisentraut R, Gunther J (1997) Individual styles of problem solving and their relation to representations in the design process. Design Studies 18:369-383

28. Akin O, Chengtah L (1996) Design protocol data and novel design decisions. In: Cross N, Christiaans H, Dorst K (eds) Analysing Design Activity. John Wiley & Sons, Chichester, UK, pp 35-63

29. Gero J, Kannengiesser U (2007) Locating Creativity in a Framework of Designing for Innovation. In: León-Rovira N (ed)Springer Boston, pp 57-66

30. Gelfond M, Lifschitz V (1988) The stable model semantics for logic programming. In: Kowalski RA, Bowen KA (eds) Proceedings of the Fifth International Conference on Logic Programming. MIT Press, pp 1070-1080

31. Barker R (1990) CASE method: entity relationship modelling. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA

32. Goel V, Pirolli P (1992) The structure of Design Problem Spaces. Cognitive Science 16(3):395-429

33. Newell A, Simon HA (1972) Human Problem Solving. Prentice-Hall, Upper Saddle River, NJ