Christopher J. MacLellan[1]
e-mail: cjmaclel@asu.edu

Pat Langley[2]
e-mail: langley@asu.edu

Computer Science,
Arizona State University,
Tempe, AZ 85281

Jami Shah
e-mail: jami.shah@asu.edu

Mahmoud Dinar
e-mail: mdinar@asu.edu

Mechanical & Aerospace Engineering,
Arizona State University,
Tempe, AZ 85281

A Computational Aid for Problem Formulation in Early Conceptual Design

Conceptual design is a high-level cognitive activity that draws upon distinctive human mental abilities. An early and fundamental part of the design process is problem formulation, in which designers determine the structure of the problem space they will later search. Although many tools have been developed to aid the later stages of design, few tools exist that aid designers in the early stages. In this paper, we describe Problem Formulator, an interactive environment that focuses on this stage of the design process. This tool has representations and operations that let designers create, visualize, explore, and reflect on their formulations. Although this process remains entirely under the user's control, these capabilities make the system well positioned to aid the early stages of conceptual design. [DOI: 10.1115/1.4024714]

Keywords: problem formulation, conceptual design, design representation, requirement analysis, computer-aided conceptual design

1 Background and Motivation

Design is one of the most complex cognitive activities in which humans engage, involving sophisticated reasoning about specifications, functional devices, and how the latter satisfy the former. As such, design has long been recognized as standing to benefit from computational aids, and there have been many success stories in the general area of computer-aided design.

However, nearly all work in this arena has focused on later stages of the design process, which involve determining the detailed structure of designed artifacts or deciding on specific values for their parameters. In contrast, the earlier, and equally important, stage of *conceptual* design, or problem formulation, has received relatively little attention. This phase plays a key role in the design enterprise, since it focuses on how one formulates the problem, which in turn constrains the alternatives considered during later stages. Thus, helping users generate promising conceptual designs early on will increase their chances of finding useful detailed designs later.

One reason is that design tasks, as typically stated by customers or marketing departments, are incompletely and ambiguously specified. To make them operational, designers must often add requirements, clarify goals, identify trade-offs, and otherwise refine the specifications they have been provided. In other cases, to make the problem solvable they may even need to reject some facets of the specification. These activities occur largely during the conceptual period, although they may well incorporate feedback from later stages, especially when the designer encounters problems that lead him to reconsider earlier choices.

There is also reason to believe that the formulation phase is the primary locus of creativity in the design process, particularly for nonroutine problems. Howard [1], in a review of both the design and psychology literature, provides evidence for this claim and identifies conceptual design and task analysis as the phases where the most creative output is produced. Furthermore, Christiaans [2] has discovered that the more time a designer spends formulating a problem, the better able he is to achieve a creative result. The reason for this relationship is straight-forward—alternative formulations of the design task can lead to radically different artifacts in the later stages. As a result, computational tools that encourage exploring the space of possible formulations should substantially increase the number and diversity of conceptual designs, which is generally viewed as desirable.

In this paper, we report a software environment to aid problem formulation during conceptual design. We start with an illustrative example that clarifies the types of decisions that we aim to support. Next, we review some basic findings about human problem solving that have informed our approach to this task. After this, we describe an innovative interactive system that assists users in conceptual design, then contrast it with earlier approaches to the same problem. In closing, we discuss limitations of our framework and suggest directions for additional research.

2 A Motivating Example

We can illustrate these ideas with an example of early conceptual design. Consider a task that involves designing a coffee grinder.

Design a device to produce ground coffee that is safe, adjustable, reusable, light weight, portable, cheap, and easy to manufacture.

These specifications are quite abstract, in that they place some conditions on the final design but also leave many aspects unstated. Moreover, the choices are far less constrained than those that arise in later stages of design, since even the basic structure of the solution is unspecified. The main challenge confronting the designer is to transform this ill-structured problem into one or more conceptual designs that respond to the task statement.

Presumably, the designer would start by translating the task specification into explicit requirements. These requirements may be functional, such as being adjustable or capable of producing ground coffee, or structural, such as being cheap and light weight. Next, he might decide that the primary function of grinding coffee should be enabled by a physical artifact or device that produces some behavior that generates ground coffee as a result. However, this statement of the design is no less abstract than the specifications they address. Thus, the designer might decide to elaborate on each of these elements in ways that provide additional details.

For example, he might choose to decompose the primary artifact into three subartifacts, a container, a stand, and a grinder,

[1]Present address: Christopher J. MacLellan, insititution: Carnegie Mellon University, e-mail: cmaclell@cs.cmu.edu.

[2]Present address: Pat Langley, institution: The University of Auckland, e-mail: p.langley@auckland.ac.nz.

Contributed by the Computers and Information Division of ASME for publication in the Journal of Computing and Information Science in Engineering. Manuscript received July 18, 2012; final manuscript received May 19, 2013; published online July 2, 2013. Editor: Bahram Ravani.

with the latter being further subdivided into subartifacts for the blade, shaft, and motor. Similarly, the designer might decompose the primary function into a number of subfunctions, such as holding the beans and breaking them into smaller parts, each supported by the subartifacts already created. He may also associate behaviors with these subdevices that clarify how each one enables some associated function. More generally, he may organize his conceptual design in hierarchical terms, with elements at lower levels providing details about how to achieve their parents.

It is important to mention that, during this elaborative process, the designer need not fixate on a single abstract solution, and that he may well note alternatives he wants to consider. For instance, he might think of two distinct ways to break coffee beans into smaller pieces, one that involves rotary grinding and another that involves lateral chopping. The designer might then note subartifacts that would enable these alternative subfunctions, as well as behaviors that describe this linkage. These disjunctive choices can be interleaved with the conjunctive hierarchical decompositions, effectively producing an AND-OR tree that characterizes the formulation.

The designer's formulation of the problem may also include issues or challenges that he has noted. These may involve specific problems that must be overcome, trade-offs that must be addressed, and other factors that may inform choices arising during the elaboration process. These are not, strictly speaking, part of the conceptual design, but they can play crucial roles in its development, and thus are worth considering as partners in the overall process.

Although much of the designer's activity during this stage involves refining artifacts, functions, and their behaviors, he may also devote some effort to altering the specifications themselves. These steps may focus on elaborating the requirements and goals given in the task description, leading both to subrequirements and subgoals that are organized hierarchically and that note alternative decompositions. However, in some cases, the designer may also retract elements stated explicitly in the specification because he decides that they are inconsistent with other elements or the trade-offs they introduce would be too difficult to achieve. Thus, conceptual design involves more than determining how to satisfy specifications; it may also involve reformulating the task along different lines.

This example demonstrates that during the problem formulation process a designer can regularly adjust his approach and use a variety of strategies. He might start by performing a requirement analysis, quickly shift to brainstorming artifacts that satisfy the structural requirements, then perform a functional decomposition, which in turn might prompt a further decomposition of the artifacts. When an impasse or dead end is encountered he might elaborate on or remove requirements before repeating the process. Any ontology or interactive tool designed to support this activity must be flexible enough to support these varying approaches.

Although all designers engage in early conceptual design and thus make decisions of the sort just described, it seems clear that some people are more effective at this stage than others, especially in terms of their ability to creatively generate many alternatives. This suggests that many designers would benefit from computational aids that help them encode, visualize, and analyze their candidate formulations. In this paper, we introduce a computational system that interactively assists designers in these tasks. This work differs from previous systems that attempt to automate the design process; our system assists the designer, but ultimately they are responsible for deciding if, and how, to respond. Before we describe this system, we should review some basic findings about problem solving that have informed it.

3 Background on Human Problem Solving

Design is naturally viewed as a problem solving activity, an area that has received substantial attention in cognitive psychology. The dominant theory of human problem solving, due to Newell and Simon [3], posits that this process involves heuristic search through a "problem space," a view that is well supported by empirical studies. However, researchers in this tradition also often distinguish between well-structured tasks and ill-structured problems [4]. Tasks of the former type may require problem solving and heuristic search, but the problems themselves are unambiguous. In contrast, ill-structured tasks have a vague character that humans must address before they can make progress.

Although designers certainly must address many well-structured tasks, we are interested here in the early stages of conceptual design, which arguably involve ill-structured tasks. These early stages revolve around problem definition and formulation, i.e., the activity of constructing a problem space rather than the activity of searching such a space (which has been more widely studied by cognitive science researchers). Most accounts of human problem solving claim that such ill-structured tasks require considerably more effort in this preliminary stage.

We can view problem formulation as closely related to the "understanding" process that arises in word problems from physics and algebra [5,6], and which even occurs in many classic puzzles [7]. Both processes transform specifications, which are sometimes ill-defined, into well-defined problem spaces that can be searched for solutions. However, these spaces do not always work out as desired, in which case the problem solver must revisit his earlier formulation. Studies of cognition on insight problems [8,9] suggest that their solution can require a change in the problem space being searched through mechanisms like elaboration, constraint relaxation, and re-encoding.

Studies of design problem solving are generally consistent with these observations. For example, Christiaan [10] presents evidence that designers who spend more time on problem definition produce better results. Fricke [11] found that the better designers spend more time on problem formulation and engage in "structured questioning" related to technical functions and attributes. Both Christiaans [12] and Atman [13] found that successful designers set their priorities early and consciously build an "image" of the problem. Valkenburg and Dorst [14] found that a more successful design team considered more framings of problems than an unsuccessful team.

Empirical studies of design also reveal issues that can arise during problem formulation. Gero [15] claims that designers interpret requirements by producing interpreted representations that they augment with implicit requirements taken from their own experiences, leading them to view the same problem differently. In some cases, this generates useful problem spaces, but Fricke [16] observed that poor designers introduced fictitious constraints and incorrect requirements that hindered their efforts to find solutions. On a related note, Harfield [17] asserts that naive designers often assume everything is fixed, whereas experts more commonly question requirements.

Although the distinction between problem formulation and problem solving has proved very useful, the former need not always precede the latter. Several researchers have reported evidence for the "co-evolution" of problem spaces and their solutions [18–20]. The general finding is that designers detect partial structures in a problem space that they use to structure their search for solutions within that space. This lets them generate initial designs, which in turn leads to restructuring of the problem space or refinements of the problem. In other words, problem solving can provide feedback that leads to improved problem formulations.

In summary, the literature on human problem solving, both in design and more generally, contains relatively few studies of problem formulation, but those which exist tell a reasonably consistent story. Problem solving can occur only after the designer has determined which space to search, which in turn depends on a process of task understanding and problem formulation. This activity plays a key role during the early stages of conceptual design, but many designers, especially novices, encounter difficulty in generating useful formulations, and even some experts do not formulate tasks as broadly as possible. Taken together, these

results buttress our claim that designers can benefit from computational aids that support problem formulation. We note that this gap in the literature does not mean an approach that focuses on problem formulation for improving design is justified; however, we do believe that the problem formulation phase merits further exploration.

4 An Interactive System for Conceptual Design

To support a designer in constructing problem formulations, we have developed Problem Formulator, an interactive design assistant. This tool focuses on the early stages of the design process and lets the designer easily input information about their conceptual designs, store this content for later review, and display it for the user's inspection. Before describing Problem Formulator's representations and operations, we should first review how we envision the system operating.

Let us return to the scenario of designing a device to grind coffee. Suppose the designer identifies some requirements for the device and enters them into the system. After providing the requirements, he thinks of functions and artifacts that would achieve them, which he also enters and links to relevant requirements. Next he generates some behaviors that characterize how the device operates and links them to related functions and artifacts. Suppose he also identifies some issues with his current design, which he also enters so that he can address them later. At the end of this process, the system informs the designer that some requirements have no links to any functions or artifacts, suggesting the design is incomplete. This prompts the user to take action to complete his formulation.

This example suggests that any interactive system for aiding conceptual design should incorporate a number of components, including

- an internal representation for encoding problem formulations;
- a graphical or textual notation for displaying a given problem formulation to the user;
- operations for creating requirements, functions, artifacts, behaviors, and issues;
- mechanisms for creating relations between pairs of entities;
- operations for editing and deleting entities and links; and
- a reasoning system for identifying problems with a design, such as incompleteness.

We believe any system that aims to aid designers in the conceptual design process should meet these six requirements. Thus, we have organized the subsections that follow in terms of how the Problem Formulator supports each of these capabilities. In some cases, we also discuss alternative approaches that we considered and rejected when developing the system.

4.1 Representation. Before our system can successfully aid the conceptual design process, it must be able to represent the types of information that designers generate. We can divide this content into two broad categories, entities and the relations between them. One complication is that often designers consider alternatives, imagining multiple internally consistent solutions to a problem that are sometimes called *protosolutions* [17]. The Problem Formulator must represent these disjunctive protosolutions in an efficient manner. To this end, the system relies on the *problem map* [21–23] ontology shown in Fig. 1 for representing conceptual designs. Because this paper focuses specifically on the tool and the ontology has been thoroughly described in previous

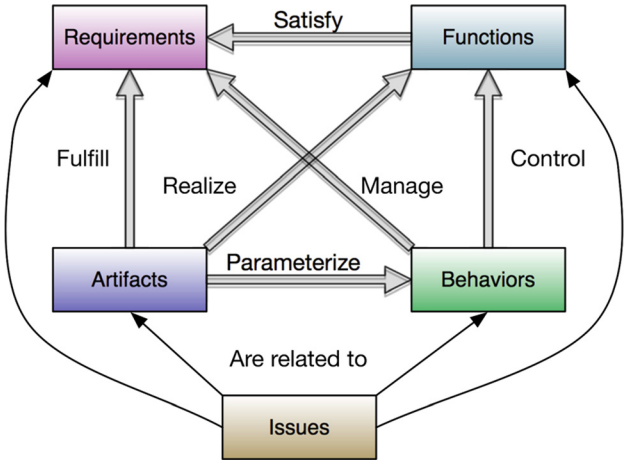


Fig. 1 The problem map ontology

papers, we will only focus on aspects of the ontology relevant to our tool and describe these aspects in the context of a running example. For a more detailed discussion of why we prefer this representation over the function-behavior-structure and structure-behavior-function ontologies, see the work of Dinar et al. [22]. To summarize this discussion, problem maps are more expressive than previous ontologies in three respects: they can model requirements, disjunctive hierarchical structure among elements in each category (e.g., among requirements), and metacognitive issues that designers may consider during the design process.

The problem map framework organizes entities into five main categories:

- Requirements, which encode the problem specification and goals for the design;
- Functions, which describe what the designed system will do;
- Artifacts, which describe structural aspects of the design;
- Behaviors, which describe how the artifact interacts with the world; and,
- Issues, which note metalevel comments about the design.

To illustrate these five categories, let us refer back to the task of designing a coffee grinder. If we focus on the single requirement of developing a device that grinds coffee, then our designer might naturally posit an abstract device or artifact that achieves this purpose. Furthermore, he might consider a function that this artifact carries out that achieve the requirement, namely grinding coffee. He might also note that this function must be governed by some behavior that transforms power input into the device into its grinding ability. Finally, he might wonder if power transformation will produce excessive heat that will have to be dealt with.

This type of conceptual thinking is easily represented in the problem map ontology, which was created for this purpose. In this case, there would be one entity for each of the five categories:

- produce ground coffee—a requirement
- grinding coffee—a function
- coffee grinder device—an artifact
- power transformation—a behavior
- will power transformation produce excessive heat?—an issue

Problem Formulator stores such content in a relational database that reflects the problem map ontology.

Requirements	Functions	Artifacts	Behaviors	Issues
Must produce ground coffee	Grind coffee	Coffee grinder device	Power transformation	Will power transformation produce excessive heat?

Fig. 2 An abstract coffee grinder design in Problem Formulator

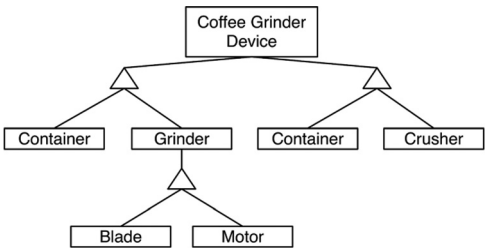


Fig. 3 Decomposition of the coffee grinder device artifact

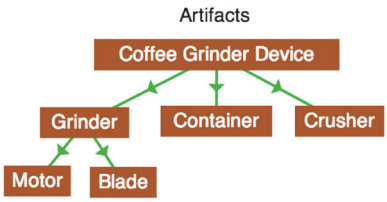


Fig. 4 Disjunctive decomposition of the coffee grinder device artifact in an earlier version of Problem Formulator

Once the system has stored this information internally, the graphical interface can display it to the user. To distinguish among the five entity types, Problem Formulator presents them in separate columns that together make up a table. Figure 2 shows a screen shot of the display for the abstract coffee grinder example.

The ability to represent individual entities, although essential to the conceptual design process, is fairly trivial; the real power comes from the ability to link these entities in interesting ways. Designers may entertain many such relationships, but we maintain they are usefully divided into intragroup and intergroup relations. The former consist of alternative sets of parent–child links between entities of the same type that specify different ways to refine the parent. The latter consists of links between pairs of entities with different types that describe how they interact.

Referring to our running example of the coffee grinder, imagine that our designer decides to decompose the grinding device into two subartifacts: a container and a grinder, where the grinder further decomposes into a blade and a motor.[1] The designer might then decide to include an alternative decomposition: a container and a crusher. We maintain that consideration of such alternatives is a central source of creativity in design.

The problem map framework lets Problem Formulator represent these parent–child relationships and their disjunctive character. Problem maps interleave entities and their decompositions in an AND-OR hierarchy that captures these connections. Essentially, each decomposition specifies a possible refinement (OR), but each such decomposition denotes a set of entities that must be achieved together (AND). In this case, Problem Formulator's graphical interface would display the functional decompositions as seven entities grouped into three sets, as Fig. 3 depicts.

The disjunctive character of decompositions makes them more challenging to visualize. Early versions of Problem Formulator showed all of the alternative decompositions at the same time, as shown in Fig. 4. However, this made the interface very cluttered, made it difficult to distinguish between alternative expansions, and appeared to discourage users from thinking disjunctively. Pilot studies suggested that designers tend to focus on only one protosolution at a time, and the current graphical interface reflects this idea.

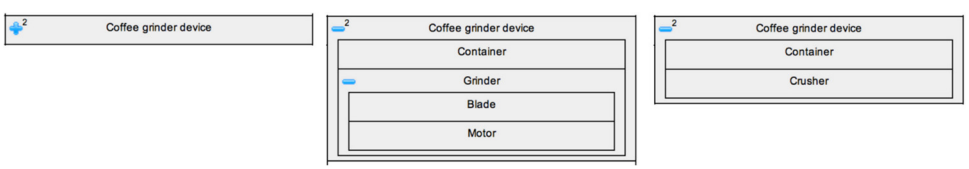
The visual metaphor that we have chosen to display these relationships in the current version is similar, in many respects, to the file/folder structure available on many operating systems, in which parents contain their children. However, the structure differs when dealing with alternative expansions because file/folder systems are not disjunctive. When expanding an entity (similar to opening a folder), Problem Formulator presents the user with a dialog box that gives alternative decompositions from which he can select. Thus, at any moment the graphical interface shows only one consistent cut through the hierarchy of decompositions.

Returning to our example, the designer might imagine alternative decompositions that realize high-level entities he has created. The designer may at one time view the problem formulation at the most abstract level (Fig. 5(*a*)), in another he may view the first decomposition (Fig. 5(*b*)), and in a third situation he may view an alternative decomposition (Fig. 5(*c*)). We believe that the construction of disjunctive parent–child relationships is one of the most important aspects of conceptual design, as it produces multiple protosolutions (internally consistent nondisjunctive designs) that might be carried to fruition.

Equally important are relations between entities of different types. These let the designer specify how requirements are satisfied, fulfilled, and managed, how functions are realized and controlled, and how issues are connected with other entities. Continuing with our example, suppose that the designer has just entered entities of different types and wants to link them. Further, suppose that he believes that the grind coffee function satisfies the requirement of grinding coffee, the coffee grinder device realizes the grind coffee function, the power transformation behavior controls the grind coffee function, the coffee grinder device fulfills the grinding coffee requirement, and the issue with whether the power transformation generates excessive heat must be linked with the suspect behavior.

Within the problem map framework, we can model this situation using a single intergroup link for each relationship that the designer wishes to represent. The problem map ontology assumes seven link types:

- artifacts realize functions
- functions satisfy requirements
- behaviors control functions
- behaviors manage requirements
- artifacts fulfill requirements
- artifacts parameterize behaviors
- issues are related to any entity type



(a) Example of no decomposition showing. (b) Example of first decomposition showing. (c) Example of second decomposition showing.

Fig. 5 Disjunctive decomposition of the coffee grinder device in the latest version of Problem Formulator

[1]Although we chose to decompose an artifact in this example, the designer might similarly decompose any of five entity types.

Fig. 6 Power transformation and its connected entities are highlighted when moused over

Therefore, the situation just discussed would be encoded with five links:

- grind coffee *satisfies* device that grinds coffee
- coffee grinder device *realizes* grind coffee
- power transformation *controls* grind coffee
- coffee grinder device *fulfills* must produce ground coffee
- will power transformation produce excessive heat? *is related to* power transformation.

Problem Formulator's graphical interface displays these links via highlighting of entities. When the designer mouses over an entity, the system highlights both it and all other entities that are connected to it, as Fig. 6 illustrates. The interface does not differentiate between different link types because only one type of relation is possible between any two types of entity. Thus, by mousing over the power transformation behavior, the user can inspect the associated links to the grind coffee function and to the issue of whether it will produce excessive heat. This somewhat ethereal approach to visualization, which reduces the clutter that results from drawing relations as lines but it still lets the designer access the desired information.

To summarize, a designer may need to represent many kinds of information during the conceptual design process. Problem Formulator divides this information into two categories: entities and relations between them. To aid a user, the system utilizes the problem map ontology to encode this content in a relational database and presents it graphically in a five-column tabular display. Problem Formulator can store multiple sets of children for each entity, but the interface lets the user inspect only one expansion of an entity at a time. In addition, when the user mouses over an entity, it highlights connected entities to denote intergroup relations. We believe that this internal structure and the associated graphical interface will help designers to organize their thoughts and to generate more insightful conceptual designs.

4.2 Operations and Mechanisms. In Sec. 4.1, we discussed how Problem Formulator uses the problem map framework to represent and display content that designers create during conceptual design. However, presumably the user does not begin with an existing problem formulation; he starts the task with a blank slate.

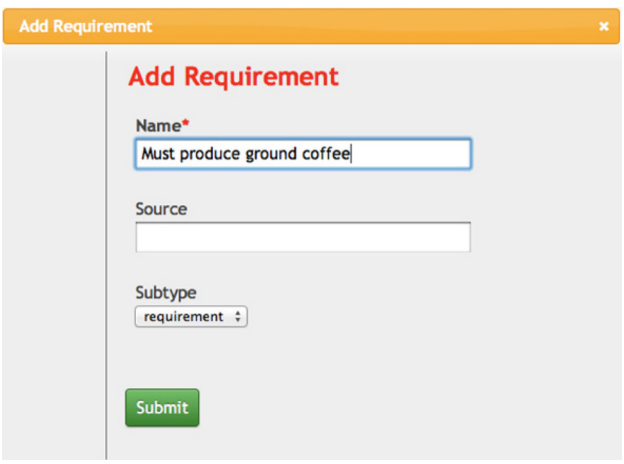


Fig. 7 The dialog that pops up after clicking the add requirement button

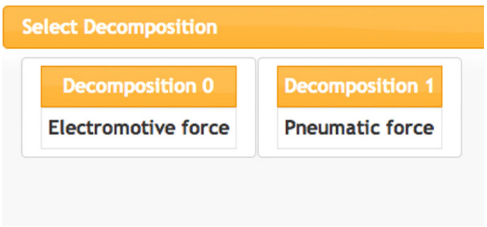


Fig. 8 The dialog that pops up after double clicking on a node with available decompositions

Thus, we now turn to the operations that Problem Formulator provides to let its users create problem maps and the mechanisms it offers for letting them explore and reflect on these formulations. As before, we illustrate each operation with an example from the task of designing a coffee grinder.

Naturally, the most basic operation needed to construct a problem map involves creating new entities, which can later be elaborated and linked. Continuing with the coffee grinder example, suppose the designer wants to construct the abstract problem map shown previously in Fig. 2. In this situation, he would add five entities, one for each group, and any associated attributes of those entities. We considered having a graphical operation for adding entities, by dragging blank entities from a sidebar into the problem formulation, but decided against it because there was no intuitive way for the user to specify entity attributes. Problem Formulator instead uses a simple dialog box for each entity type (see the dialog for requirements in Fig. 7) that lets users specify any optional or required attributes. In our pilot studies, this approach seemed to be effective.

Once a designer has interactively populated the problem map with some initial entities, he can begin to link them. For example, suppose he wants to decompose the power transformation behavior in two alternative ways: in terms of either electromotive force or pneumatic force. Let us assume the designer has already added the behavior for the first decomposition (electromotive force) and wants to state that it is a sub-behavior of power transformation.

As things stand, the power transformation behavior has no decompositions, so the user must create the new decomposition and the first parent–child link. We might have performed this operation with another dialog box in which the user specifies the parent-child relationship, but in a pilot study with a previous version of the tool (which used a dialog box) subjects commented on how slow and tedious this approach was. Therefore, Problem Formulator adopts a more intuitive approach in which the user links a child entity like electromotive force into the parent entity. If the parent currently has no expanded decomposition, the system responds by creating a new decomposition and adding the child to it. The user can then add a second child to the same decomposition by dragging it into the parent while the previous decomposition is still expanded.

Now suppose that, having established a first decomposition, the designer decides to add the second decomposition, pneumatic force. Using the same graphical approach, he can collapse the

Fig. 9 The designer dragging the grind coffee function into the must produce ground coffee requirement to create a link

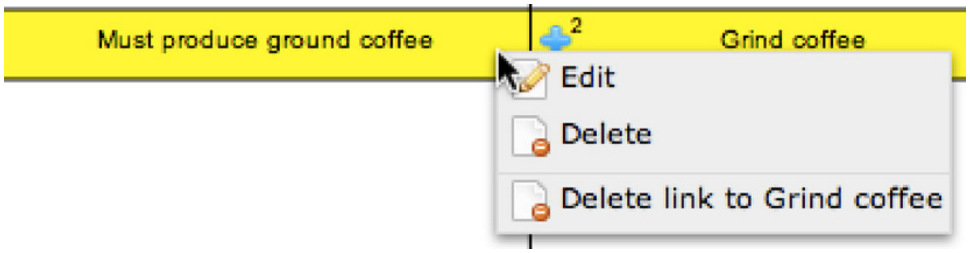


Fig. 10 The context menu that pops up when the designer right clicks on an entity

power transformation behavior (currently expanded) back to a more abstract level. The user accomplishes this by a simple interface action such as double clicking on the entity, thus collapsing and hiding its children. He then creates the pneumatic force behavior, after which he drags it into the parent entity. Since the parent would be collapsed, Problem Formulator responds by creating a new decomposition and adds the child entity to it. As before, the user can then add more children by dragging them into the parent while the decomposition is expanded.

After creating multiple decompositions, the designer might want to switch between them. Because Problem Formulator already utilizes graphical operations to create links, we decided to use similar operations to switch between decompositions. In particular, the system lets the designer collapse an entity with a simple interface action (e.g., double clicking) and re-expand it with the same action, at which point it presents him with alternative decompositions which he can select from (see Fig. 8). By requiring the user to collapse and re-expand entities to select among decompositions, we maintain our commitment to showing only one expansion at a time while still making it easy to switch between alternative designs. We believe this is a useful feature of any conceptual design assistant.

Once the user has added entities to some of the groups, he might then decide to link entities of different types to indicate how they are related. Returning to the coffee grinder, suppose that the designer has an abstract problem map with only one entity in each group (shown in Fig. 2) and that he wants to link the grind coffee function with the requirement for a device that grinds coffee. Although we could have made the user create such links through a dialog box, we decided that a graphical operation would be more intuitive, so we incorporated one into Problem Formulator; subjects from our pilot studies subsequently had no problems with this approach. To link two entities, the user drags one of them (in this case, the function) into the other (in this case, the requirement), as shown in Fig. 9. This action is the same for all link types because Problem Formulator automatically determines the link type for any pair of entities.

After the designer has created some entities and linked them, he may decide to edit or delete some elements. For example, he might determine that the requirement "device to grind coffee" might be better stated as "device to produce ground coffee." To support such basic changes, we decided to incorporate a context menu, with the Problem Formulator user opening the context menu on the relevant entity (by an action such as right clicking) and selecting "edit." This action reopens the dialog shown during entity creation, letting the designer revise any of the attributes. If he wants to delete an entity, then he opens the context menu and selects "delete," as shown in Fig. 10.

In addition to editing and deleting entities, the designer might decide to remove some relations he has established. Following the conventions already introduced, he can easily remove parent–-child links by dragging a child entity out of the parent, much like moving a file out of a folder. If a decomposition contains no more children, then Problem Formulator automatically deletes it and returns the parent to the collapsed state. Similarly, the user can easily eliminate intergroup relations by opening the context menu and selecting the option to delete the intended link. Using Problem Formulator for the coffee grinder task, the designer might produce a problem map similar to the one shown in Fig. 11.

Once the designer has used the Problem Formulator interface to produce a problem map, he can also take advantage of mechanisms for exploring and reflecting on his creation. The first of these lets the user view connections between entities of different types. Viewing such intergroup relations may be problematic because one of them may be hidden under a collapsed parent. To deal with such situations, Problem Formulator highlights the connected entity and its parents in successively lighter shades (Fig. 12). Thus, when a linked entity is hidden, its parent will appear in a lighter shade of highlighting, informing the user that one of its children is connected to the entity currently being moused. To find the connected entity, the user single clicks on the entity he is mousing over, causing all highlighting to freeze, then expands the lightly shaded parent. In response, all of the parent's decompositions will appear, but only the one containing the connected entity will be shaded. The designer can follow this highlighting until he finds the connected entity.

Problem Formulator also aids designers by providing a basic search mechanism. One can easily imagine a situation in which

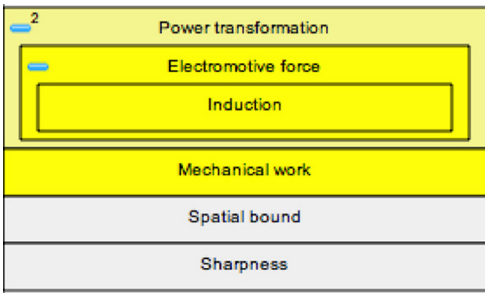


Fig. 11 The electromotive force, induction, and mechanical work behaviors and their parent entities highlighted in successively lighter shades when a connected entity has been moused over

Fig. 12 The do not operate when unsealed requirement and its parent entities highlighted in successively lighter shades by the search mechanism

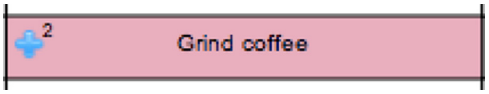


Fig. 13 The grind coffee entity is highlighted with red by Problem Formulator because it is currently unrealized by any artifact, implying that the user should connect it

the user remembers creating an entity but, having encoded multiple disjunctions, is unsure what entities he must expand to find it. In such cases, he can start to type the entity's name in the search bar and the interface will automatically complete its name and highlight the entity in the graphical display (Fig. 13). If the entity is not currently visible, then its parents will be highlighted in successively lighter shades. As the designer expands entities that are lightly shaded, the highlighting will continue to appear in alternative decomposition's dialogs, making it easy for the user to track down the missing entity. When done, he clears the search term and Problem Formulator removes the highlighting.

A final Problem Formulator feature, fundamentally different from the others, aids designers in reflecting on their problem formulation throughout the conceptual design process. This mechanism uses metalevel knowledge to detect incompleteness in a problem map. The system incorporates three generic rules[2] to this end:

- no entity should be entirely unconnected;
- every requirement should be satisfied by a function or fulfilled by an artifact; and
- every function should be realized by an artifact.

After each user interaction, Problem Formulator invokes a logical reasoning module that applies these rules to the current problem map. Upon finding an entity that violates one or more rules, the system highlights the entity in red, thus informing the user that he may want to attend to it (see Fig. 14), much as in Kassoff's [24] work on highlighting inconsistencies in logical spreadsheets. Once the user corrects the problem, the annotation vanishes. This highlighting, while unobtrusive, lets the designer quickly focus on and repair those aspects of the problem map that are incomplete.

In summary, Problem Formulator includes interactive operations and mechanisms that let its users construct, explore, and reflect on problem maps. These include operations for adding entities, creating parent–child links, relating entities of different types, and editing or deleting entities and links. Mechanisms for exploration and reflection let the user view links at different levels of abstraction, search for entities in the abstraction hierarchy, and identify incomplete problem formulations using meta-level reasoning. The latter has potential to support the design process in additional ways, the most obvious being the enumeration of all protosolutions entailed by a problem formulation.

4.3 System Architecture. We have implemented Problem Formulator in a manner that lets users access it from the World Wide Web. We had two reasons for this choice. First, making the software available on the Web makes it more accessible; users can run it from any location and on any device that operates with a modern Web browser (no additional software needs to be installed on the user's computer). Second, all entities and links that the user enters are stored in the cloud, where they are backed up and easily retrieved for future use, regardless of location or device. To provide Problem Formulator with these features, we utilized CakePHP—a model-view-controller (MVC) framework—along with a combination of PHP, JavaScript, MYSQL, HTML, CSS, and Answer Set Prolog. We chose CakePHP because the MVC framework makes the software more modular and easier to develop and maintain. We chose Answer Set Prolog to support logical reasoning because it is fast, flexible, and reliable.

Figure 15 shows the four components that make up Problem Formulator: the stored problem formulation, the controller, the view, and the inference backend. The system encodes problem formulations internally in the problem map ontology, which, as explained earlier, consists of different entities and relations among them. Problem Formulator stores this content in the relational database structure shown in Fig. 16. The view determines how it displays information stored in the problem map to the user. There are views for all basic functions, such as adding and deleting entities and links, as well as ones for the user's active projects. The controller determines what information from the problem map is available in each view. There are controllers for creating and

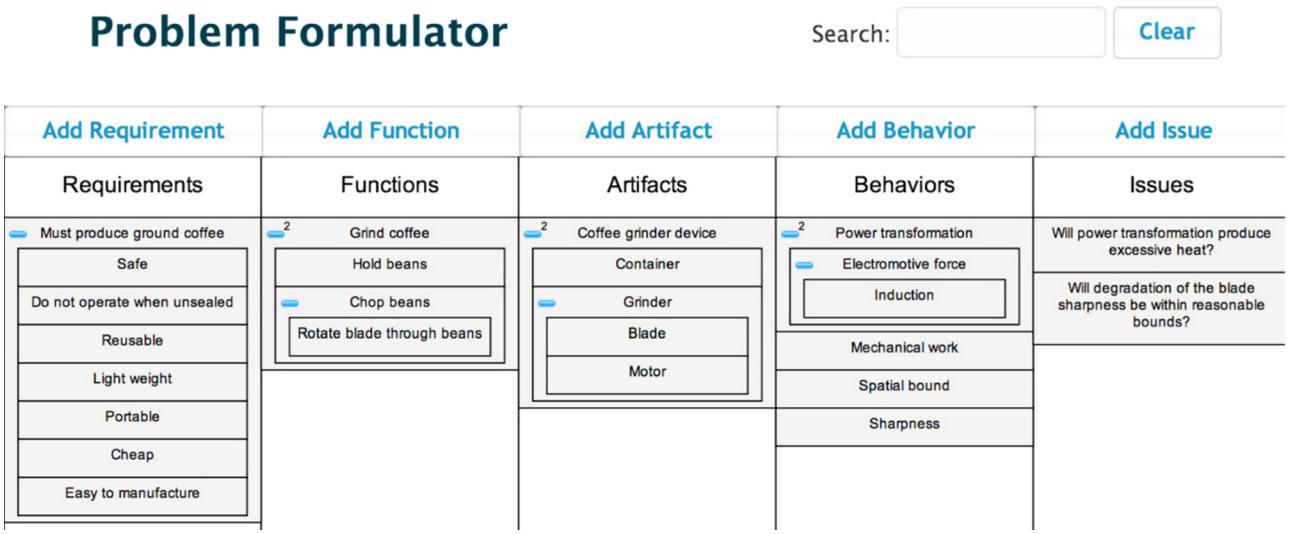


Fig. 14 A screen shot of the Problem Formulator tool with the completed coffee grinder example

[2]Although these rules are reasonably simple, one can imagine more sophisticated variants that could provide even more useful suggestions to designers, but the latter would remain responsible for deciding whether, and how, to respond.

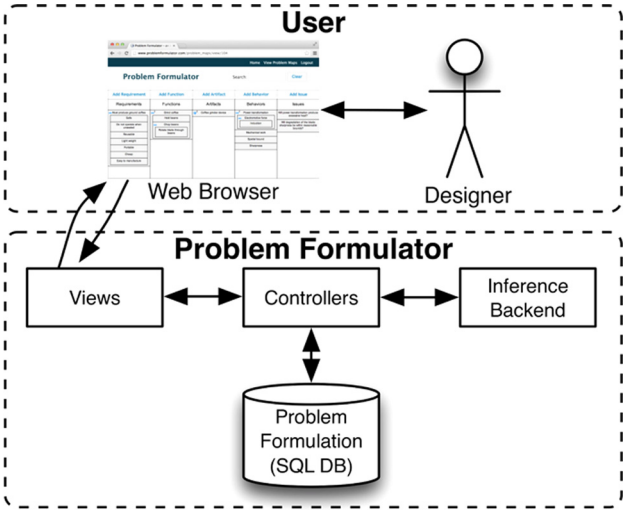


Fig. 15 System diagram for Problem Formulator

manipulating problem formulations, entities, and links; these provide a layer of abstraction between the problem map model and the view that ensures data integrity. Finally, the inference backend incorporates logical reasoning methods that let Problem Formulator identify incomplete problem formulations.

The designer connects to Problem Formulator through a Web browser, which displays their problem maps. When a user takes action in their browser, the changes to the problem map are sent to the server where they are stored in the database. Meanwhile, the interface is dynamically updated using Javascript, so the user never has to refresh their browser. If Problem Formulator generates any feedback for the user, then it is sent to the web browser, which dynamically updates the problem map with the feedback.

5 Comparison to Other Interactive Aids

There has been relatively little research on interactive tools that aid the early stages of conceptual design, with most work focusing on later stages in the design process. However, there have been some efforts on related topics, which we review here.

For example, there has been a substantial body of work, both commercial and open source, on interactive software for concept mapping [25], but this has not focused on engineering design. As a result, these systems do not take advantage of the structures that arise in design domains, such as the distinction between requirements, functions, artifacts, and behaviors. In addition, they generally lack the ability to represent disjunctive choices of any kind.

A different area of research that does focus on engineering design concerns interactive software for sketching. Systems in this paradigm [26,27] provide a flexible means for users to construct, store, and visualize sketches of designed artifacts. However, they offer no support for situations in which the user does not yet know what to sketch, which we maintain is crucial to early conceptual design. This relates to their emphasis on artifacts rather than on requirements, functions, or behaviors.

A third related research theme, on ideation tools [28–30], aims to help designers overcome impasses they encounter during their design activities. These toolkits attempt to stimulate their user via various ideation techniques that are supposed to overcome mental blockages. Such methods are certainly relevant to early conceptual design, but the software systems in which they are embedded typically lack any graphical interface for creating or visualizing explicit problem formulations.

A fourth area connected to our research involves systems for product life-cycle management [31,32], which stores information about products and lets multiple users access and update this content. Although one might use such systems for conceptual design, they are seldom used for this task [33], presumably because they are inefficient for this purpose. Additionally, even though these systems incorporate version control, they are not ideal for recording the intense cognitive processes on which designers rely during early phases of design.

Another system similar to ours is FunctionCAD [34], which uses the functional basis ontology [35] to allow users to model the functional aspects of their designs. Our system builds on these ideas to allow designers to model additional aspects of their designs (i.e., requirements, artifacts, behaviors, and issues), to represent disjunctive hierarchical decompositions of entities in each category, and to model connections between entities in different categories. Unlike our work, research in this area has placed a great emphasis on the development of a common database of functional information, which can be used for a wide range of purposes, such as automated concept generation [36]. We have instead focused on how to support users in the absence of domain knowledge. The goals are different, but complementary; one could

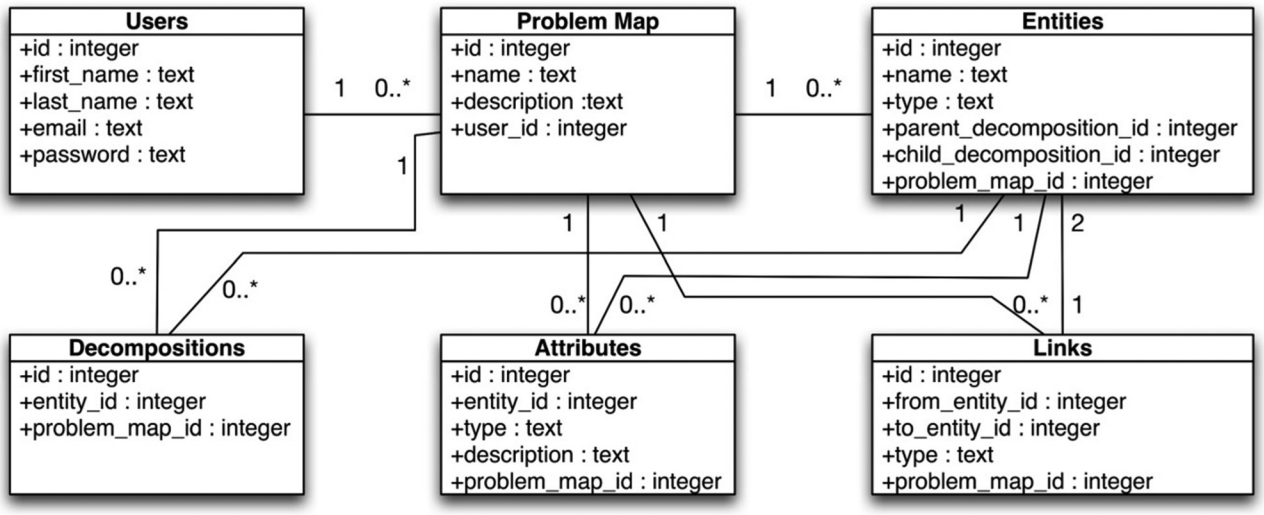


Fig. 16 Database schema for Problem Formulator

imagine a system that aids users by providing knowledge about the domain and supports them in creating and reflecting on conceptual designs.

Finally, the closest relative to our approach is SBFAuthor [37], an interactive environment for constructing problem formulations stated in the structure-behavior-function ontology. This system lets users create and visualize the structures, behaviors, and functions of a conceptual design, and its authors even discuss using model checking to identify incompleteness in a given formulation, although they did not appear to implement this idea. However, SBFAuthor lacks the ability to represent disjunctions and it does not include either requirements or issues, all of which we view as central to conceptual design. Nevertheless, this work has similar aims to ours and has influenced the development of our approach.

In conclusion, although there exist a wide range of computational tools available for use by designers, there are few that focus on aiding early conceptual design, and even fewer that provide interactive support for the construction of problem formulations during this crucial stage. To our knowledge, Problem Formulator is the only system that addresses all of the major issues by providing the ability to represent and visualize the entities and relations that arise during conceptual design, as well as operations for interactively constructing, exploring, and reflecting on problem formulations.

6 Directions for Future Work

Although Problem Formulator system shows considerable promise for aiding early conceptual design, it also has clear limitations that we should remedy in future research. One drawback is that the graphical interface does not provide the user with information about how well their problem formulation fares along dimensions related to creativity. Examples include the total number of protosolutions that can be derived from the problem map and the diversity of those alternatives. Calculating such measures and incorporating them into the display (e.g., using color coding) would tell users which areas of their conceptual design have been fleshed out sufficiently and which would benefit from additional effort. Experts might well ignore some of these cues when they are confident that they have covered enough territory, but novices could gain substantially from such attentional assistance.

Another area for improvement involves adding the ability to ground the problem map in protosolutions, each of which is a consistent set of functions, artifacts, behaviors, and specifications. The great strength of the problem map notation is that it can encode many different designs in an efficient, implicit manner, but in some cases the user may desire to inspect the particular alternatives that his problem map represents. For this reason, we plan to extend Problem Formulator to itemize all such protosolutions for inspection by the user. In some cases, the user may desire to see not a complete design but rather a subdesign that corresponds to a particular node in the problem map. We also intend to explore methods for grouping protosolutions and displaying them in a taxonomic hierarchy, with more similar candidates being closer to each other.

A third limitation is that the current system can make only a few suggestions to users. Problem Formulator can use meta-level checking rules to detect incompleteness in the current problem map (e.g., due to functions not enabled by artifacts), but it only highlights the nodes involved. An improved system would incorporate generative variants of these meta-level rules that, whenever they note an incomplete situation, augment the problem map with an issue and even introduce a "skolem" node that the user can edit to resolve the problem. More advanced versions could utilize meta-level rules that recommend more sophisticated activities. For instance, the system might encode a strategy of breadth-first expansion of alternatives that it could invoke when it observes the user pursuing a depth-first strategy of drilling down into details. One can imagine a variety of such high-level strategies that could encourage more creative designs.

Naturally, we should also evaluate both the basic Problem Formulator system and its extensions with human subjects. Studies should include both novice and expert designers, as well as ones identified in advance as being more or less creative. In general, we expect novices and less creative individuals to receive the greatest benefit, but we also expect others to appreciate many of its features. We should also extend Problem Formulator to record the details of user interactions for use in later analysis. These may reveal insights about the strategies that distinguish more productive designers from others, which in turn we can incorporate into more advanced versions of the system.

7 Concluding Remarks

In this paper, we presented an innovative computational approach to assisting the early stages of conceptual design. We linked this activity to problem formulation, as opposed to problem solving, since it produces the problem space to be searched during later phases. We described Problem Formulator, a web-based system that provides users with a graphical interface through which they can create and visualize a problem map.

Such a problem map comprises a set of entities, including requirements, functions, artifacts, behaviors, and issues, as well as relations between them. Entities may be organized in a hierarchy, with lower levels providing more details about the conceptual design. A problem map may also include multiple expansions that reflect alternatives the user wants to consider. Together, these decisions imply a set of protosolutions, each corresponding to a consistent conceptual design.

Problem Formulator system incorporates operations for creating new entities, for deleting ones no longer desired, and for linking pairs of entities through a variety of relations. The interface lets users visualize the current problem map in a tabular format, with one column for each entity type. Hierarchical expansions are shown within embedded boxes that the user can collapse when he desires a more abstract view. Problem Formulator also includes checking rules that let it detect and highlight incomplete facets of the problem map that require user attention.

We found that Problem Formulator differs radically from earlier systems that support concept mapping, sketching, and ideation, in that it provides a structured representation well suited for engineering design, supports disjunctive alternatives that encode multiple protosolutions, and incorporates a graphical interface for entering and visualizing design content. We do not claim that these features make Problem Formulator superior to existing tools; instead, we claim that they are features that would be valuable to any system that assists users in creating and reflecting on their conceptual designs. Finally, we noted that the current version of Problem Formulator has clear limitations, but also that it suggests a number of natural avenues for extension. As such, the Problem Formulator offers a promising approach to aiding problem formulation during the key stage of early conceptual design.

Acknowledgment

This study was supported by the National Science Foundation, CMMI Grant No. 1002910. The opinions expressed in this paper are those of the authors and are not endorsed by the National Science Foundation. We thank Kurt Vanlehn for advice and feedback on the development of the system and Glen Hunt for his work on initial versions of Problem Formulator.

References

[1] Howard, T. J., Culley, S. J., and Dekoninck, E., 2008, "Describing the Creative Design Process by the Integration of Engineering Design and Cognitive Psychology Literature," Des. Stud., **29**(2), pp. 160–180.

[2] Christiaans, H. H. C. M., 1992, "Creativity in Design: The Role of Domain Knowledge in Designing," Ph.D. thesis, Delft University of Technology, Lemma, BV.

[3] Newell, A., and Simon, H. A., 1972, *Human Problem Solving*, Prentice-Hall, New York.

[4] Simon, H. A., 1974, "The Structure of Ill Structured Problems," Artif. Intell., **4**(3), pp. 181–201.
[5] Chi, M. T. H., Feltovich, P. J., and Glaser, R., 1981, "Categorization and Representation of Physics Problems by Experts and Novices," Cogn. Sci., **5**, pp. 121–152.
[6] Hinsley, D. A., Hayes, J. R., and Simon, H. A., 1977, "From Words to Equations: Meaning and Representation in Algebra Word Problems," *Cognitive Processes in Comprehension*, P. A. Carpenter and M. A. Just, eds., Lawrence Erlbaum Associates, Hillsdale, pp. 89–106.
[7] Hayes, J. R., and Simon, H. A., 1974, "Understanding Written Problem Instructions," *Knowledge and Cognition*, L. W. Gregg, ed., Lawrence Erlbaum, Oxford.
[8] Ohlsson, S., 1992, "Information-Processing Explanations of Insight and Related Phenomena," *Advances in the Psychology of Thinking*, M. T. Keane and K. J. Gilhooly, eds., Harvester Wheatsheaf, London.
[9] MacLellan, C., 2011, "An Elaboration Account of Insight," ACS: Papers From the AAAI Fall Symposium, pp. 194–201.
[10] Christiaans, H. H. C. M., and Dorst, K., 1992, "An Empirical Study Into Design Thinking," *Research in Design Thinking*, N. Roozenburg and K. Dorst, eds., Delft University Press, Delft.
[11] Fricke, G., 1999, "Successful Approaches in Dealing With Differently Precise Design Problems," Des. Stud., **20**(5), pp. 417–429.
[12] Christiaans, H. H. C. M., 1992, "Cognitive Models in Industrial Design Engineering: A Protocol Study," *Design Theory and Methodology*, D. L. Taylor and D. A. Stauffer, eds., American Society of Mechanical Engineers, New York, pp. 131–140.
[13] Atman, C. J., Chimka, J. R., and Bursic, K. M., 1999, "A Comparison of Freshman and Senior Engineering Design Processes," Des. Stud., **20**(2), pp. 131–152.
[14] Valkenburg, R., and Dorst, K., 1998, "The Reflective Practice of Design Teams," Des. Stud., **19**(3), pp. 249–271.
[15] Gero, J. S., 2004, "The Situated Function-Behaviour-Structure Framework," Des. Stud., **25**, pp. 373–391.
[16] Fricke, G., 1993, "Empirical Investigation of Successful Approaches When Dealing With Differently Precised Design Problems," ICED, pp. 359–367.
[17] Harfield, S., 2007, "On Design 'Problematization': Theorising Differences in Designed Outcomes," Des. Stud., **28**(2), pp. 159–173.
[18] Cross, N., and Dorst, K., 1998, "Co-Evolution of Problem and Solution Spaces in Creative Design: Observations From an Empirical Study," *Computational Models of Creative Design IV*, J. S. Gero and M. L. Maher, eds., University of Sydney, New South Whales.
[19] Maher, M. L., and Poon, J., 1996, "Modeling Design Exploration as Co-Evolution," Comput.-Aided Civil Infrastruct. Eng., **11**(3), pp. 195–209.
[20] Kolodner, J., and Wills, L. M., 1996, "Powers of Observation in Creative Design," Des. Stud., **17**(4), pp. 385–416.
[21] Dinar, M., Shah, J. J., Hunt, G. R., Campana, E., and Langley, P., 2011, "Towards a Formal Representation Model of Problem Formulation in Design," IDETC/CIE, pp. 1–10.
[22] Dinar, M., MacLellan, C., Danielescu, A., Shah, J. J., and Langley, P., 2012, "Beyond Function-Behavior-Structure," DCC, J. S. Gero, ed.
[23] Dinar, M., Shah, J. J., Langley, P., Hunt, G. R., and Campana, E., 2011, "A Structure for Representing Problem Formulation in Design," ICED, S. J. Culley, B. J. Hicks, T. C. McAloone, T. J. Howard, and W. Chen, eds., pp. 392–401.
[24] Kassoff, M., Zen, L.-M., Garg, A., and Genesereth, M., 2005, "PrediCalc: A Logical Spreadsheet Management System," VLDB, pp. 1247–1250.
[25] Novak, J., and Cañas, A. J., 2008, "The Theory Underlying Concept Maps and how to Construct and Use Them," Technical Report IHMC CmapTools 2006-01 Rev 2008-01, Institute for Human and Machine Cognition, Pensacola, Florida.
[26] Dorta, T., 2008, "Ideation and Design Flow Through the Hybrid Ideation Space," Rev. Centro de Invest. Univ. La Salle, **8**(29), pp. 25–30.
[27] Gross, M. D., 1996, "The Electronic Cocktail Napkin—A Computational Environment for Working With Design Diagrams," Des. Stud., **17**(1), pp. 53–69.
[28] Mohan, M., Shah, J. J., and Narsale, S., 2012, "Capturing Ideation Paths for Discovery of Design Exploration Strategies in Conceptual Engineering Design," DCC, J. S. Gero, ed.
[29] Mohan, M., 2011, "A Framework for Holistic Ideation in Conceptual Design Based on Experiential Methods," Ph.D. thesis, Arizona State University, ProQuest UMI.
[30] Zlotin, B., Zusman, A., Altshuller, G., and Philatov, V., 1999, "Tools of Classical TRIZ," Technical Report No. 266, Ideation International Inc., Ideation International Inc.
[31] Sääksvuori, A., and Immonen, A., 2008, *Product Lifecycle Management*, Springer-Verlag, Berlin.
[32] Bergsjö, D., Almefelt, L., Dinar, M., and Malmqvist, J., 2010, "Customizing Product Data Management for Systems Engineering in an Informal Lean-Influenced Organization," Syst. Res. Forum, **4**(1), pp. 101–120.
[33] Jennings, M., and Rangan, R., 2005, "Managing Complex Vehicle System Simulation Models for Automotive System Development," ASME J. Comput. Inf. Sci. Eng., **4**(4), pp. 372–378.
[34] Nagel, R. L., Perry, K., Stone, R. B., and McAdams, D. A., 2009, "FunctionCAD: A Functional Modeling Application Based Design Framework," IDETC/CIE.
[35] Stone, R. B., and Wood, K. L., 2000, "Development of a Functional Basis for Design," ASME J. Mech. Des., **122**, pp. 359–370.
[36] Bryant, C., McAdams, D. A., Stone, R. B., Kurtoglu, T., and Campbell, M., 2005, "A Computational Technique for Concept Generation," IDETC/CIE.
[37] Goel, A. K., Rugaber, S., and Vattam, S., 2009, "Structure, Behavior, and Function of Complex Systems: The Structure, Behavior, and Function Modeling Language," Artif. Intell. Eng. Des. Anal. Manuf., **23**(1), pp. 23–35.