First Annual Conference on Advances in Cognitive Systems Poster Collection (2012) 19-34

A Model of Problem Formulation Strategies in Engineering Design

Mahmoud Dinar MDINAR@ASU.EDU
Jami J. Shah JAMI.SHAH@ASU.EDU
Mechanical and Aerospace Engineering, Arizona State University, Tempe, AZ 85287 USA

Abstract

Different designers use different strategies to formulate a design problem. We introduce the Problem Map framework to study these differences, with formalized statements of formulation strategies. To formalize the strategies we define as set of operators in the Problem Map framework and represent formulation states. We have identified five tasks of problem formulation. For each task, we give an example of how a strategy can be traced in changes among states or in the sequence of the operators that were used to reach the resulting state. We also give examples of traces of three different strategies in protocols collected from eight experienced designers.

1. Introduction

Designers are known to have individual styles of solving design and non-design problems (Eisentraut 1999). One way of characterizing the differences between designers is to find the strategies they adopt during designing. We are interested in modeling the differences among designers' strategies in problem formulation which is an understudied aspect of design thinking.

An inspiring field of work is model tracing in intelligent tutoring systems (Anderson et al. 1990). In intelligent tutoring systems, models for knowledge acquisition of an expert and a student are compared to prompt students with the steps they should take in solving a problem. Though the problems that are taught in such systems are usually well-defined problems (e.g. math, physics, or diagnosing a failure in a machine), we can draw analogies in modeling and tracing evidences of certain strategies in solving or formulating design problems. To trace design strategies, we should employ a state-operator model that captures the intense cognitive processes of design.

An important aspect that we consider in modeling design strategies is domain independence. This is because we can state meta-level rules that govern design strategies in a general way with a domain independent framework. We use the Problem Map framework (Dinar et al. 2012), to represent design strategies, since it is domain-independent, and we can define a state-operator model in it.

A motivating example of a design strategy is whether designers abstract or specify an aspect of a problem definition. When exploring the design space, a designer can add more detail to an idea, or generalize that idea. The ability of abstracting concepts is considered one of the divergent thinking skills that designers may possess to different degrees (Shah et al. 2012). To see whether

a designer has employed an abstraction or specification strategy in an interval, we can examine the changes in two state models at the beginning and the end of that interval, and see if the designer added more specific details to a stated thought, or if he generalized those parts to more abstract concepts.

2. A Framework for Representing Design Strategies

We have developed a framework for studying the relation between problem formulation and creative outcome. The fine level of detail of the framework enables us to capture more of the intense cognitive processes that underlie early conceptual design. We have used Problem Maps to represent the differences among different designers in formulating a problem (Danielescu et al. 2012). In order to explain how we can use Problem Maps to state problem formulation strategies, we briefly introduce the elements of the Problem Map framework and its representations.

2.1 The Framework

The *Problem Map* framework has five groups of entities. Each group has a base entity, and a few groups have supporting entities. The entities have optional attributes. Each group has a common hierarchical structure and groups are interrelated. Disjunctive decompositions and relations are allowed within and between groups respectively. The framework is represented in Figure 1.

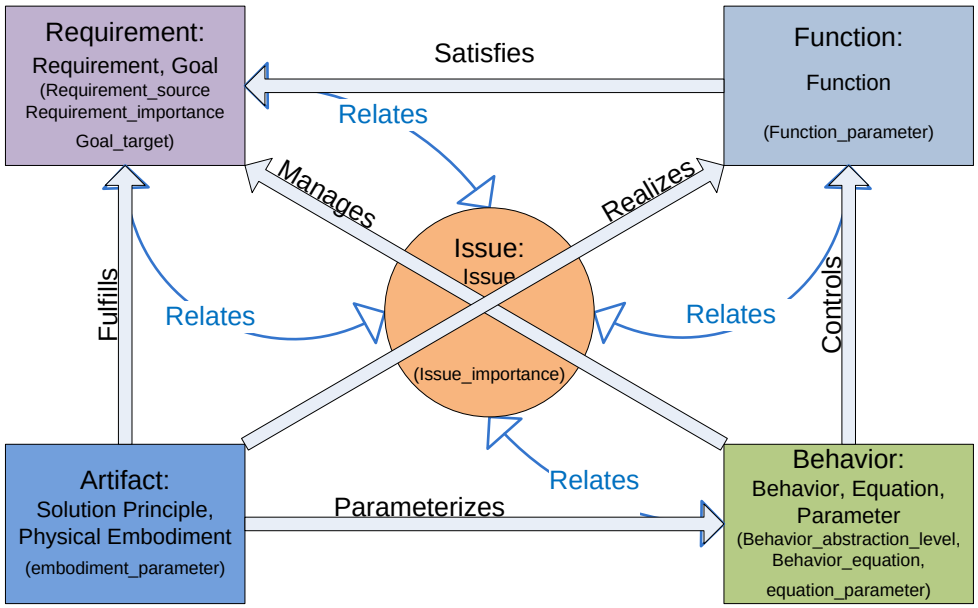


Figure 1. The Problem Map Framework: entities and intergroup relations.

A Problem Map is a set of states which contain instances of the aforementioned entities with the different possible relations they may have with each other. We can consider the initial state as

the one with all the given requirements. Since designers do not follow a prescribed or a specific method of design or problem formulation, many possibilities may ensue for the next state.

Though we do not have a strict definition of what a formulation state is, consider the simple case where any change, such as the addition of a new instance of an entity, specifying an attribute of an existing entity, or relating two instances, alters the current state. **Error! Not a valid bookmark self-reference.** depicts these operators. There are five high-level operators with which we determine the low-level operators from their corresponding set. The operators act on instances which are the arguments within the braces in the sets. An example of a high-level operator is "Specify attribute", while "Specify requirement source" attributing source "sc" to requirement "rq" is a low-level operator. The "Intra-relate" operators are populated by the five groups of entities. The parent and the child, or the before and the after instances are from the same entity group. The hierarchy attribute denoted by "hy" is an identifier which can show alternative decompositions. Different decompositions may have common entities and thus one parent-child or before-after relation can be common in alternative decompositions; hence the "hy" identifier. Alternative inter-group relations do not have common entities. The set of alternatives for the "Propose" operator is written as a set of the two previous sets for a compact representation. It does not mean that the operator is a combination of other operators. Similarly, the "Delete" operator acts on data that is added with the other operators.

Table 1. The operators of the Problem Map

Add entity{requirement(rq),function(fn),artifact(ar),behavior(bh),issue(iu)}
Specify attribute{requirement_source(rq,sc),…,equation_parameter(bh,qp)}
Intra-relate structure{parent-child(pt,ch,hy),before-after(bf,af,hy)}
Inter-relate relation{satisfies(fn,rq),realizes(ar,fn),…,relates(iu,any)}
Propose alternative{structure{}, relation{}}
Delete data{entity{},attribute{},structure{},relation{}}

The number of low-level operators is about forty. From our past experience with analyzing protocols with P-map models, at least five hundred of such operations occur in an hour-long design session. This is equivalent of a branching factor of 40^{500}. To find certain patterns (of specific sequences) among different designers, an exhaustive search might not be the best way. We can define design strategies and trace their occurrences in searching for specific changes in the formulation states.

2.2 Formalizing a Strategy Within Problem Maps

We can trace strategies by comparing two states in an interval that we expect the strategy to be employed. Consider the example of abstraction vs. specification strategies that we gave in the introduction. To trace instances of these strategies, we look for the states which include parent-child relations. Then we locate the two states that contain the parent, and the child. If the state that has the parent occurs after the state that has the child, it indicates that the designer followed an abstraction strategy.

To state the abstraction strategy formally, we can use the Problem Map operators that we explained in the previous section. The strategy may be employed with any entity but suppose an example with requirements. Three operators in a specific order define the strategy. Two are the addition of the requirements and one is relating the parent to the child where the parent was added before the child; see **Error! Not a valid bookmark self-reference.**.

Table 2. A model of the abstraction strategy for requirements

Add requirement(rq1) **Add** requirement(rq2) **Intra-relate** parent-child(rq2,rq1,hy1)

3. Modeling problem formulation strategies

We have come up with a list of strategies that designers may exhibit during designing. Some of these are based on the literature in design, and some are hypothesized introspectively. We have grouped them into five tasks that we identified in problem formulation, see Table 3.

For each task, we give an example of one strategy. We state each strategy in a neutral sentence, without specifying whether it is representative of good or bad design practice, simply because there is no hard evidence for or against them. One of the main objectives of our research is to find the unknown effect of problem formulation strategies. To do the study, we should trace evidences of designers implementing the strategies, and to do so we should state these strategies formally.

3.1 Tasks of Problem Formulation

Suppose that the initial state is where the designer is briefed about the problem statement. This can be shown by a set of given requirements depicted by R_g as in Figure 2. In the following figures we show different states that explain each strategy by changes to the initial state. We should emphasize that the examples that we give for each strategy are not the only possible states that occur following the specified strategies. In addition, the states that we sometimes refer to as final states, do not necessarily represent when problem formulation or design ends. Design problems, unlike well-defined problems such as puzzles, do not have clear termination criteria, or parsimonious processes.

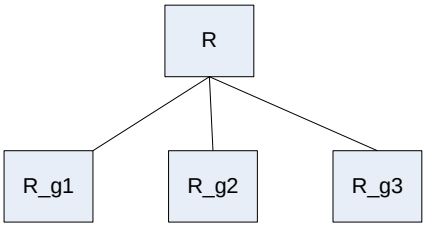


Figure 2. The initial state of a problem map with given requirements

Table 3. A list of design strategies for the five tasks of problem formulation

Task	Strategies
Information gathering	Good designers are goal driven; they ignore irrelevant data Novices are data driven; they try to make something of every piece of information in the design brief
Perception and assumption	Good designers treat problems as ill-defined; they continuously add/delete requirements Poor designers introduce fictitious constraints and incorrect requirements Good designers identify the most critical requirements and focus on those first, ignoring routine requirements Expert designers rely on abstract analogies while novices analogize surface-level cues
Problem decomposition	Good designers create multiple formulations Scoping is an important part of problem formulation – reducing design space Some designers are problem-driven; they develop requirements and functions before working on artifacts and behaviors Some designers decompose the problem to sub-problems first, and elaborate on artifacts and behaviors for each sub-problem
Augmentation and support	Good designers create and use multiple representations to discover gaps, inconsistencies and conflicts Good designers ask a lot of questions Designers use metaphors that abstract and map some aspect of the design to other domains to gain an understanding Designers analogize by different aspects of the source device: the function, the product architecture, the behavior, and the issues
Analysis and verification	Experienced designers use more generative reasoning, in contrast to the deductive reasoning used by inexperienced designers Good designers can identify conflicting requirements and functions Designers reflect on their process, what they did, did not do; they set priorities, and determine an efficient greedy task sequence Some designers propose artifacts and examine their behaviors to see if they meet the requirements

3.1.1 Information Gathering

For information gathering, we model the following strategy:

> Some designers are goal-driven; they ignore irrelevant data.

Designers gather information from different sources such as catalogs during designing, but in the early stages, i.e. problem formulation, we can assume that the main source is the problem statement, either as a written brief, or from an interview with the customer. In P-maps, this information is in the given requirements. The strategy of ignoring irrelevant information can be stated in P-maps as, not all given requirements are followed. In other words, there will remain a

few given requirements that are not decomposed further, or related to other entities. Figure 3 shows a state where the designer defined functions for one given requirement and one derived requirement, and followed up the formulation by proposing an artifact and specifying its behavior for one of the functions.

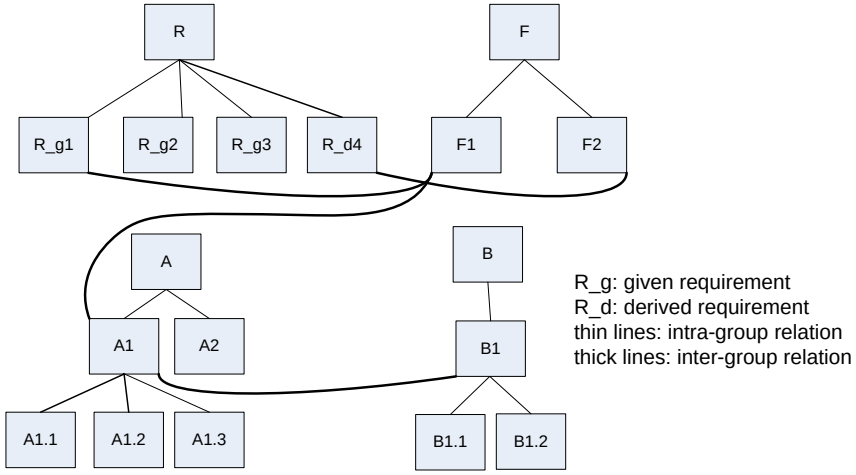


Figure 3. An example of the state after following the information gathering strategy

Modeling this strategy implies actions or operators that do not occur. However, we can still model this strategy without using negation by stating only the operators that are used. We trace the strategy in the absence of the operators that would otherwise prove that it was not employed. We can also determine the degree to which working on a requirement constitutes lack of attention to another requirement. This can be going from one requirement to a function, artifact, and a behavior, in addition to decomposing one or more of these related entities further for at least one requirement as in Figure 3. The operators for this definition are stated in Table 4. Notice that for simplicity, we only show the relevant operators. The absence of "Inter-relate" or "Intra-relate" operators for two of the given requirements is evidence that the defined strategy was followed.

Table 4. The operators for the given example of information gathering strategy

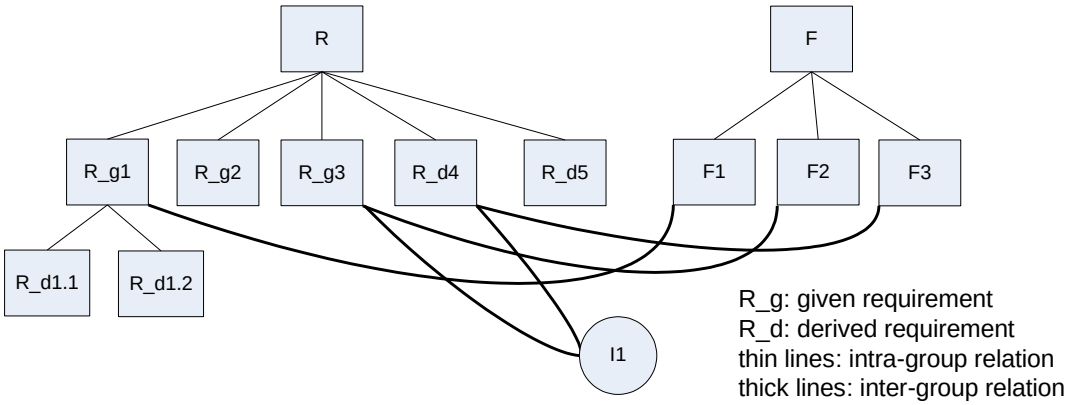
Add requirement(R_g1) … **Add** requirement(R_d4)
Specify requirement_source(R_g1,given)
Specify requirement_source(R_d4,derived)
Inter-relate satisfies(F1,R_g1)
Inter-relate realizes(A1,F1)
Inter-relate parameterizes(B1,A1)
Intra-relate parent-child(A1,A1.1,hy_A1)
Intra-relate parent-child(B1,B1.1,hy_B1)

3.1.2 Perception and Assumption

The next strategy is about how designers perceive a problem after they gather information about it. We model the following strategy:

> Some designers deliberately treat problems as ill-defined and question given requirements.

We described the information gathering strategy in terms of which sources of information would be used. The perception and assumption strategy is about judging the credibility and the importance of the designer's assumption about the problem. We can state this in P-maps as less number of requirements are given, that they are added throughout the design session, and that some given requirements may be dropped. An example of this strategy is shown in Figure 4. It implies dropping or altering some requirements after they were elaborated. The operators corresponding to this example are given in **Error! Not a valid bookmark self-reference.**. For simplification, we assume referential integrity where the deletion of an entity means the deletion of all data regarding that entity.



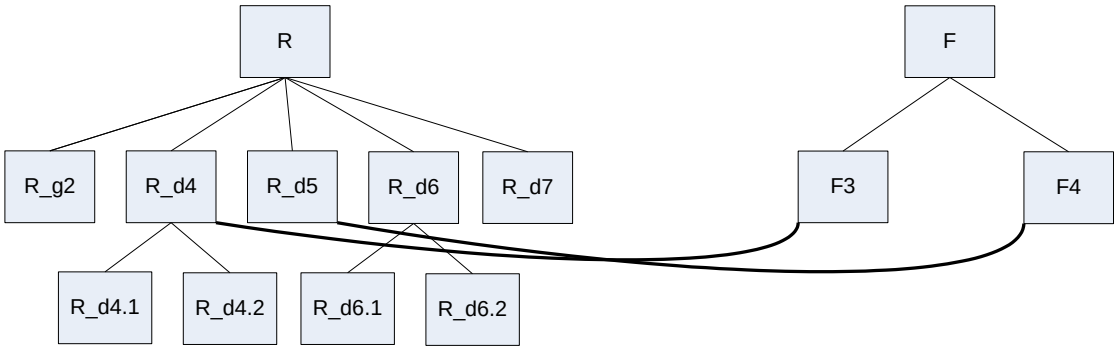


Figure 4. An example of the states that represent the perception and assumption strategy

Table 5. The operators for the given example of perception and assumption strategy

Add requirement(R_g1) **Specify** requirement_source(R_g1,given) **Add** requirement(R_d1.1) **Specify** requirement_source(R_d1.1,derived) **Intra-relate** parent-child(R_g1,R_d1.1,hy_R1) **Inter-relate** satisfies(F1,R_g1) **Delete** requirement(R_g1) **Add** requirement(R_d6)… **Add** requirement(R_d6.2) **Specify** requirement_source(R_d6.2,derived)

3.1.3 Problem Decomposition

For the next task, problem decomposition, we model this strategy:

> Some designers are problem-driven rather than solution-driven; they focus on understanding the problem before attempting to solve it.

In P-maps we can see evidence of adopting this strategy when designers add artifacts and behaviors, or add more details to them towards the end of the session. Similarly to the previous strategy, the example is best understood in the transition from the initial state to the point where we search for the strategy through an intermediary state; see Figure 5. Requirements are elaborated in both states, but the elaboration of artifacts and behaviors appear at the final state. Additionally, hierarchies, especially in requirements and functions are expanded in breadth first at each level. The operators of this example are shown in **Error! Not a valid bookmark self-reference.**. The operators that add new entities at lower levels appear after all entities were added at the level above.

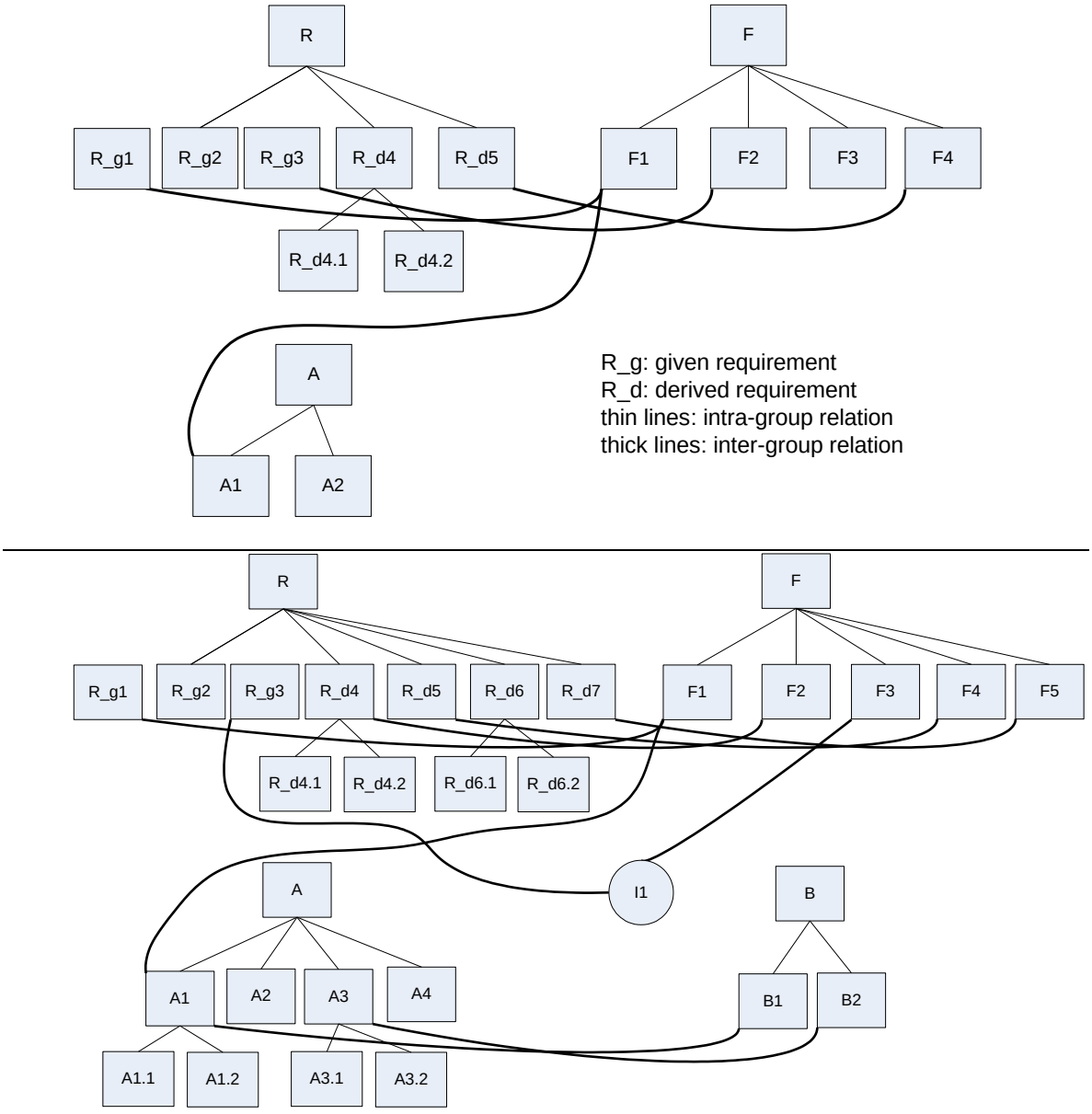


Figure 5. An example of the states following the problem decomposition strategy

Table 6. The operators for the given example of problem decomposition strategy

Add requirement(R_g1) … **Add** requirement(R_d5) **Add** requirement(R_d4.1), **Add** requirement(R_d4.2) **Intra-relate** parent-child(R_d4,R_d4.2,hy_R1) **Add** function(F1), **Inter-relate** satisfies(F1,R_g1) **Add** function(F4), **Inter-relate** satisfies(F1,R_d5) **Add** artifact(A1), **Inter-relate** realizes(A1,F1) **Add** requirement(R_d6) … **Add** requirement(R_d7) **Add** requirement(R_d6.1), **Add** requirement(R_d6.2) **Intra-relate** parent-child(R_d6,R_d6.2,hy_R1) **Add** function(F5), **Inter-relate** satisfies(F5,R_d7) **Add** function(F4), **Inter-relate** satisfies(F1,R_d5) **Inter-relate** relates(I1,R_g3), **Inter-relate** relates(I1,F3) **Add** artifact(A4) **Add** artifact(A1.1), **Add** artifact(A3.2) **Intra-relate** parent-child(A3,A3.2,hy_A1) **Add** behavior(B1), **Add** behavior(B2) **Inter-relate** parameterizes(B1,A1) **Inter-relate** parameterizes(B2,A3)

3.1.4 Augmentation and Support

We have chosen the following example of an augmentation and support strategy to model:

> Some designers generate multiple and more abstract representations to discover inconsistencies and build insight.

In P-maps this strategy can be seen as a combination of different actions. One is manifested in generating more disjunctive decompositions. Another will be that, requirements and functions are decomposed more frequently than artifacts or behaviors are. The other is that more solution principles are used rather than physical embodiments, and artifacts in general are added frequently without being further decomposed or related to many entities. An example of the resulting state and the operators corresponding to it are given in **Error! Not a valid bookmark self-reference.** and Table 7.

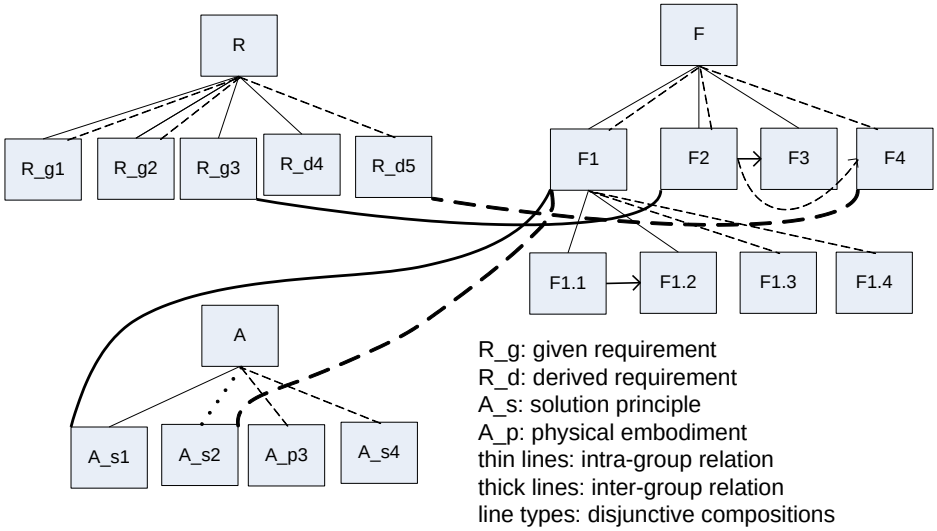


Figure 6. An example of the resulting state of the augmentation and support strategy

Table 7. The operators for the given example of the augmentation and support strategy

Add requirement(R_g1) … **Add** requirement(R_d4)
Intra-relate parent-child(R,R_g1,hy_R1)…
Intra-relate parent-child(R,R_d4,hy_R1)
Add requirement(R_d5)
Propose parent-child(R,R_g1,hy_R2)
Propose parent-child(R,R_d5,hy_R2)
Intra-relate before-after(F2,F3,hy_F1)
Intra-relate parent-child(F1,F1.1,hy_F1)
Intra-relate parent-child(F1,F1.2,hy_F1)
Intra-relate before-after(F1.1,F1.2,hy_F2)
Intra-relate before-after(F2,F4,hy_F2)
Intra-relate parent-child(F1,F1.3,hy_F2)
Intra-relate parent-child(F1,F1.4,hy_F2)
Add artifact(A_s1), **Inter-relate** realizes(A_s1,F1)
Add artifact(A_s2), **Propose** realizes(A_s2,F1)
Add artifact(A_p3), **Add** artifact(A_s4)
Intra-relate parent-child(A,A_s1,hy_A1)
Propose parent-child(A,A_p3,hy_A3)
Propose parent-child(A,A_s4,hy_A3)

3.1.5 Analysis and Verification

For the last task, we model this strategy:

> Some designers identify key issues, especially in conflicting requirements, and set priorities.

We can show this in P-maps with more issues that are added. These issues are sometimes in relation with two or more requirements, reflecting conflicts that the designer detects. Another indication of following the strategy is in further decomposition of the issues. Figure 7 and Table 8 show the graphical representation of the resulting state and the operators that reveal when this strategy was implemented.

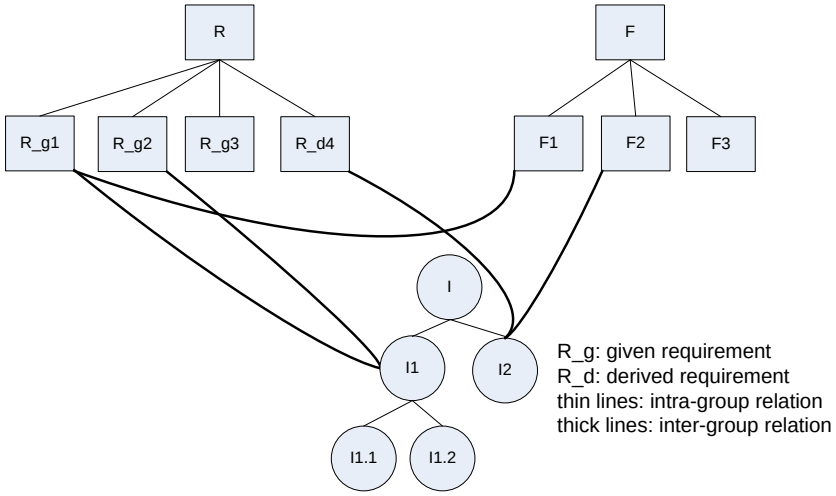


Figure 7. An example of the state showing the analysis and verification strategy

Table 8. The operators for the given example of the analysis and verification strategy

Add requirement(R_g1) … **Add** requirement(R_d4)
Add issue(I1)
Inter-relate relates(I1,R_g1)
Inter-relate relates(I1,R_g2)
Add function(F2)
Add issue(I1)
Inter-relate relates(I2,R_d4)
Inter-relate relates(I2,F2)
Intra-relate parent-child(I1,I1.1,hy_I1)
Intra-relate parent-child(I1,I1.2,hy_I1)

3.2 An Operational Example

We do not have a fully functioning system that detects when the strategies that we exemplified are employed yet. However, we have a working example of how it can be operationalized. Previously, we had collected protocols from eight experienced designers working on designing a water sampler. We encoded the protocols as P-maps (Danielescu et al. 2012). We then defined three strategies and searched for their traces in the encoded protocols.

We presented the first strategy as an introductory motivation. We called it upward abstraction. We only found abstractions for functions and artifacts. For example, one designer first proposed using a sandbag for submerging the device and shortly afterwards thought more generally of using a consumable. In another example, a designer thought of a function that could cancel the sampling operation, and added it as a parent to the ascending function.

For the second strategy, we thought that some designers might follow a strict process, for example in moving from requirements to functions rather than other entities. We called it forward processing. We looked at each requirement to see if it was satisfied by a function before being related to other entities. Only two designers had found relations between a requirement and a function, and of the two, only one had defined this relation before relating the requirement to entities other than function.

The third strategy was about whether designers add entities similarly to depth-first or breadth-first search. We called this strategy breadth-first decomposition. We could not expect the designers to strictly expand each branch before adding other branches to the hierarchy. Thus we looked instead for the number of times the depth-first expansion was violated. For each parent-child relation with the same parent, we looked for the number of times when the child became a parent itself before all the relations for the first parent were subsumed. We found no examples of this strategy among our eight subjects. The results of tracing these strategies are given in Table 9.

Table 9. Number of occurrences of three strategies for eight experienced designers

Designer / **Strategy**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Upward abstraction	3	7	4	11	4	3	12	5
Forward processing	0	0	1	0	0	0	0	0
Breadth-first decomposition	0	0	0	0	0	0	0	0

4. Other Representations of Design Thinking Strategies

Looking at the literature, two topics are of interest. First is a descriptive model of design processes. The models of activities and their representations may suggest how to formulate strategies. The strategies can be formulated directly in terms of specific sequences of actions, or indirectly by looking at the changes in the state of the mental models that represent design thinking. Second, observations of good and bad designing practices may suggest what the strategies should or should not promote. Knowing how the strategies should be stated, and what they should be about is necessary for finding problem formulation strategies that lead to creative designs.

A few past studies of design thinking have described models of the design process, parts of which relate to design strategies and their representations. These models were inspired by Newell & Simon's (1972) study of human problem solving with state-operator sets. Similarly, they employed protocol analysis as the method for data collection and analysis. Two notable efforts were carried out by Ullman, Dietterich, & Stauffer (1988), and Gero & Mc Neill (1998).

Ullman et al. (1998) pioneered modeling design cognition in the field of mechanical engineering. In their Task Episode Accumulation model, the design state consisted primarily of proposals and constraints. They defined ten operators such as *select*, *calculate*, and *reject*. They called a meaningful sequence of the operators to address some primitive goals an episode. They identified six types of episodes: *assimilate*, *plan*, *specify*, *repair*, *verify*, and *document*. They also stated that within each episode, designers choose the operators based on a set of heuristic rules. Finally, at the highest level, different episodes compose a design task. A task is a stage of the design process, e.g. conceptual, or detail design.

The observations from Ullman et al. (1998) came from analyzing both the operators and the states. They calculated the amount of time that the subjects spent on each operator, episode, and task. For example, at the conceptual design stage, they observed that the subjects spent most of the time on assimilating information and specifying proposals. Similarly, they calculated the number of constraints and proposals, and found for example that derived constraints increase significantly during post conceptual stages, while given constraints mostly appear at the conceptual stage.

Another established framework is Function-Behavior-Structure (Gero 1990). Gero described the activities in the design process in terms of transformations between pairs of function, behavior, structure, and the documented design. Gero & Mc Neill (1998) identified a set of micro and macro strategies based on the sequences and the iterations of how functions, behaviors, and structures were encoded from protocols.

Other studies, especially those of the differences between expert and novice designers, give some clues about the strategies that designers use. However, seldom are they defined in a general way, independent of the task, or adopted deliberately, much less tested to see how they influence creativity. Ho (2001) stated that expert designers approach directly the main goals and work backward for required knowledge. He added that novice designers eliminate parts of the problem when they fail to handle it. Ball & Christensen (2009) described mental simulation and analogizing as strategies that designers use when faced with uncertainty. Edelman, Agarwal, Paterson, Mark, & Leifer (2012) introduce the idea of scaffolds as vehicles to build insight, which can be of metaphors that abstract and map some aspect of the design to other domains. They find that radical breaks come from generating many alternatives, often with frequent redesigns.

Most of these strategies are based on observations with the objective of understanding the design process. There are no explicit hypotheses about these strategies to check when they occur or under what condition. There is also a need to study the effect of these strategies on design outcome.

5. Conclusions and Future Work

We introduced the Problem Map framework to represent problem formulation strategies in design. We defined the operators in the framework, and explained how a certain combination of different operators, often in specific sequences, account for a design strategy. We defined the five

tasks of problem formulation, and gave an example of a formalized strategy for each task, in terms of possible changes from an initial state, and the set of operators that are involved in forming that strategy. We also gave examples of tracing three different strategies in collected protocols. It is likely that some of the strategies that we want to trace later occur infrequently or even not at all.

We have built a computer tool based on the Problem Map framework that enables us to collect massive data from many designer subjects. Armed with the knowledge of pertinent problem formulation strategies, we can examine how effective it is to encourage designers to follow the strategies that we can correlate with creative outcome. We can use established measures in the field of engineering design to determine which outcomes are more creative. We will be able to instill the correlations that we find, as a recommendations system that is embedded in our tool. For each relevant design strategy, we can write the advisory rules that recommend that strategy, and specify the situation where it applies. Then, we can see whether prompting designers with a certain strategy will improve their creativity or not. The tool facilitates a convenient way of looking for instances of the strategies automatically, for a large number of participants in the learning phase, and later on in the testing phase of the research. A rigorous method of hypothesis testing may ultimately turn design into the science of design.

Acknowledgements

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 should also thank Pat Langley and Christopher J. MacLellan for their previous contributions to our project.

References

Anderson, J. R., et al. (1990). Cognitive modeling and intelligent tutoring. *Artificial Intelligence, 42,* 7-49.

Ball, L. J., & Christensen, B. (2009). Analogical reasoning and mental simulation in design: Two strategies linked to uncertainty resolution. *Design Studies, 30,* 169-186.

Danielescu, A., et al. (2012). The structure of creative design: What problem maps can tell us about problem formulation and creative designers. *Proceedings of ASME DETC*. Chicago, IL.

Dinar, M., et al. (2012). Beyond function-behavior-structure. *Proceedings of the Fifth International Conference on Design Computing and Cognition*. Texas A&M University, College Station, Texas: Springer.

Edelman, J., et al. (2012). Understanding radical breaks. In H. Plattner, C. Meinel, & L. Leifer, (Eds.), *Design Thinking Research: Studying Co-Creation in Practice*. Berlin: Springer Berlin Heidelberg.

Eisentraut, R. (1999). Styles of problem solving and their influence on the design process. *Design Studies, 20,* 431-437.

Gero, J. S. (1990). Design prototypes: a knowledge representation schema for design. *AI Magazine, 11,* 26-36.

Gero, J. S., & McNeill, T. (1998). An approach to the analysis of design protocols. *Design studies, 19,* 21–61.

Ho, C. (2001). Some phenomena of problem decomposition strategy for design thinking: Differences between novices and experts. *Design Studies, 22,* 27-45.

Newell, A., & Simon, H. (1972). *Human problem solving*. Upper Saddle River, NJ: Prentice-Hall.

Shah, J., et al. (2012). Applied tests of design skills - part 1: Divergent thinking. *Journal of Mechanical Design, 134,* 1-10.

Ullman, D., Dietterich, T., & Stauffer, L. (1988). A model of the mechanical design process based on empirical data. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 2,* 33–52.