INCREMENTAL DESIGN REVISION IN BIOLOGICALLY INSPIRED DESIGN

A Dissertation
Presented to
The Academic Faculty

by

Bryan Joseph Wiltgen

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in Computer Science in the
School of Interactive Computing

Georgia Institute of Technology
December 2018

INCREMENTAL DESIGN REVISION IN BIOLOGICALLY INSPIRED DESIGN

ACKNOWLEDGEMENTS

While I literally wrote this dissertation, I know now that a dissertation does not happen in a vacuum. I am grateful to the numerous supportive people (my parents, friends, and mentors) that have been part of my life during this process. You all helped me to grow as a scientist and, more importantly, as a person. Thank you all. For everything.

To Dr. Ashok Goel, my Ph.D. advisor: your attention, guidance, and unending patience as I struggled and stumbled to the finish line meant the world to me. I am lucky to call you my advisor, and I would not be where I am without your efforts.

To my committee (Dr.'s Nancy Nersessian, Mark Riedl, Spencer Rugaber, and Jeannette Yen): thank you all for your patience and support. You all stuck with me through long silences and well past a normal graduation timeline. Thank you for being there for me and for encouraging me onwards. I owe Spencer Rugaber a special debt of gratitude, for he was part of co-developing the SBF* language used in this dissertation and was a helpful presence in my research throughout the time I spent working on my degree.

To my parents, Dolleen and Timothy Wiltgen: you never once gave up on me, even when I was on the verge of giving up on myself. Words cannot express how much I treasure your support and presence in my life. Thank you so much. I love you both.

Thank you to Dr.'s Julie Linsey and Marc Weissburg from Georgia Tech. You took time out of your busy schedules to help me construct the gold standard models. I really appreciate the time and energy that you gave me to do that.

To my friends that I made while in Georgia: thank you for being my support group for so many years. Swaroop Vattam and Michael Helms, beyond being my friends, you two were my mentors. You showed me the ropes and helped me become the scientist I am today. Thank you. Keith McGreggor, Maithilee Kunda, Jim Neidhoefer, Jay Blumling, David Joyner, and several others: thank you so much for your friendship and support. I won't forget any of you.

Near the end of this process, I went to work for IBM Watson Health. To my team there, especially my managers, team leads, and close teammates: thank you from the bottom of my heart for your support and leniency while I completed my degree. You all expressed a kindness and support that means so much.

To all of my friends: you all are awesome. I am a better person for having known you, and your friendship and support means more to me than you know.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Design is the process by which solutions get developed to solve social challenges, and its products can be seen across our world from toothbrushes to computers to spaceships. Conceptual design is an early phase of design where an initial candidate solution gets developed. Analogical design is a form of design where knowledge from some known source case is transferred into the design solution, leveraging knowledge not directly related to solving the problem at hand. Biologically inspired design is a real-world form of analogical design where source cases come from nature.

In all of this, there is a need for design revision. The output (the candidate design solution) likely begins in an imperfect state and needs to be revised to ensure it properly and completely solves the design problem. Additionally, the designer's understanding of any source cases or prior solutions used could also exist in an imperfect state and should too be revised since they are used in reasoning. Thus, any flaws in this knowledge could negatively impact the process and/or outcome. This is especially important in an interdisciplinary context like biologically inspired design where a given design practitioner may have expertise in only some of the involved domains.

To support design revision, this dissertation presents an AI agent that can help a design practitioner engage with a process of incremental design revision. In this hypothesized, iterative process, a practitioner externalizes their knowledge of a design concept in a functional model, gives this model to the AI agent that evaluates the model, and then receives feedback from the agent. The practitioner can then decide to revise their model based on this feedback, and then can restart the cycle or conclude it.

Specifically, this dissertation addresses the task of model evaluation to support incremental design revision in the context of the conceptual phase of biologically inspired design. It presents the Design Evaluation through Simulation and Comparison (DESC) AI agent, a computational approach to evaluate the candidate design, biological source cases, and prior designs in this context. More precisely, DESC evaluates functional model articulations of these concepts in the form of Structure-Behavior-Function Star (SBF*) models. These functional models act as proxies for the designer's understanding or vision of the modeled designs.

DESC does its evaluation through two techniques: Simulation and Comparison. Simulation evaluates the behaviors (processes or mechanisms) and functions (intended or perceived purposes) of a model by simulating the behaviors and determining if those results conflict with claims made in the model. This checks the internal consistency of the model–the extent to which the model's parts agree with each other–for the language rules understood by the simulator. Comparison evaluates the complete model (i.e., the structure submodel, the behaviors, and the functions) by comparing it to another model of the same topic and identifying differences, which are potential areas of misunderstanding or misrepresentation. This checks the external consistency of the model–the extent to which it accurately represents a concept in the world or (in the case of the candidate design) a proposed vision. As part of the Comparison technique, this dissertation also explores a novel analogical mapping algorithm called Compositional Mapping for matching elements between two models.

This dissertation evaluates DESC in two ways. First, experimentation was done in the form of a computational ablation-like study to illustrate DESC's ability to evaluate models using Simulation and Comparison. The results support the hypotheses that these techniques can evaluate the internal and external consistency of models and that the SBF* modeling language has syntax and semantics that support automated simulation. This computational experimentation also partially supported Compositional Mapping's advantage over a more conventional approach to mapping models for the purposes of Comparison.

The second form of evaluation was a pilot study with human participants that tested the usefulness of DESC in helping humans construct better models and improve their understanding of a design concept thematically related to biologically inspired design. In other words, it explored at a high level to what extent DESC can support the hypothesized incremental design revision cycle. The results of this study partially supported the hypothesis that DESC can support incremental design revision, showing that participants with DESC tend to produce superior models compared to those without it. However, this did not translate to improved understanding.

To recap, this dissertation presents an AI agent called Design Evaluation through Simulation and Comparison (DESC) for evaluating designs in the form of SBF* functional models, contextualized in a hypothesized incremental design revision process in the conceptual design phase of biologically inspired design. Evaluation of this AI agent supports that it can indeed evaluate models and partially supports its usefulness in supporting incremental design revision, with analysis that suggests people with an implementation of DESC produce better models than those without it. The overall evaluation of DESC through the computational experimentation and the usefulness study presents promising results towards the broader incremental design revision goal.

CHAPTER I

INTRODUCTION

This dissertation focuses on incremental design revision in the context of the conceptual phase of biologically inspired design. Design is a common human reasoning activity where an individual or team develops a solution, typically an artifact, to resolve some problem or challenge. The products of design can be seen everywhere in human experience from one's toothbrush to spaceships. Given its significance, supporting design is inherently motivating.

Analogical design is a kind of design that transfers knowledge from a known source case (or cases) to the design solution, typically without directly using the source case as a solution. Biologically inspired design (BID) is a kind of analogical design where practitioners[1] take inspiration from nature to address their design challenges, producing creative and, sometimes, sustainable design solutions. This paradigm enables novel approaches to challenging design problems, leveraging the natural world as a library of source cases. BID is thus a socially meaningful endeavor that represents a real-world context for studying design.

Finally, conceptual design is the process by which a practitioner defines the set of functions (or goals) that are needed to resolve the design problem and develops the structures that will achieve those functions, leading to a candidate design. In so doing, the practitioner also may engage with knowledge of prior designs and, in the case of BID, biological source cases. For simplicity, each of the candidate design, a prior design, or a biological source case will here be generically referred to as a design.

Practitioners reason using internal models of these designs. The accuracy of these models is important for the success of the process. The candidate design is the output of

[1]The word practitioner is used in this dissertation as a synonym for designer or modeler to reduce the overuse of design/model terms.

conceptual design, and getting it correct builds a strong foundation for future stages. The prior designs and source cases are inputs to the process. Prior designs may be directly used in solving the design challenge, or they may form the starting point by which, through analogy, the practitioner derives (either all of part of) the candidate design. Knowledge is transferred by analogy from the biological source case and used in the development of the candidate design. Thus, the accuracy of one's knowledge of the biological system is important, and zooming out, the accuracy of these three types of designs will impact the process.

Vattam et al. [161] motivate some of this context. Regarding the need to develop one's understanding of a biological source: "Developing a biologically inspired design solution involves retrieving a suitable biological system, *understanding how that system works to a sufficient degree of depth*, extracting mechanisms and principles associated with that system into a solution-neutral form, and applying those mechanisms and principles in the target domain of engineering" (emphasis added). They also identify an example that shows the consequence of not knowing enough about the source. In a design case studied in this work, the authors imply that at least part of the reason the design team did not choose to use mangrove roots as their biological source case was because "[n]ot enough was known about mangrove roots." How might this design have gone had the authors a tool that could help them incrementally revise their knowledge? Finally, the authors also provide motivation for evaluating the candidate design. They describe how the same design team went to an expert for feedback on their conceptual design. The authors write, "[t]he expert suggested that their initial design would not work," highlighting, in the language of this dissertation, that the candidate design was imperfect according to this expert.

Assume that a practitioner's internal model of a design is imperfect, perhaps due to misconceptions or lack of understanding. Imperfect knowledge is plausible in the biologically inspired design context because of its interdisciplinary nature. Although a given practitioner may have expertise across all the domains engaged by a design challenge, it is possible that a practitioner may be an expert in biology but not in engineering, or vice versa. Thus, a practitioner may have inaccurate knowledge of the biological source case (biology) or a prior design (engineering). Regarding the candidate design, the practitioner's knowledge may be imperfect because the concept is still in development (e.g., the candidate design is in its early stages). An imperfect model implies there could exist a better model. Given the need for design accuracy, one goal then is to revise the design towards increasingly better models. If transitioning from the initial model of the design to the final version is a journey rather than a single step, the process of achieving this change can be thought of as a sequence of incremental design revisions.

This dissertation also introduces the complementary goal to develop an artificial intelligence (AI) agent that supports designers. Such an agent, an artificial teammate so to speak, could help the human practitioner do incremental design revision in a systematic and repeatable manner, both of which are enabled because the agent can address revising in a deliberate way that is executed the same each time.

This thinking raises two questions:

1. **What does it mean for one version of a design to be better than another?** Answering this question operationalizes what is meant by revising and guides how one should effect desirable change in internal design models.

2. **How might an AI agent support systematic and repeatable incremental design revision?** Answering this question is the thrust of this dissertation.

In answering how to support incremental design revision, a strategy was devised that the AI agent would take as input an articulation of the practitioner's internal design model and produce as output a set of potential issues that the practitioner should investigate. This feedback from the agent provokes incremental design revision and creates a hypothesized cycle: (1) practitioner updates (or creates the initial version of) their internal design model, (2) the AI agent evaluates an externalization of that model, (3) the practitioner reflects on the evaluation feedback, and then the cycle repeats if desired. Figure 1.1 depicts this hypothesized process.

Such a high-level strategy raises a third question to be answered.

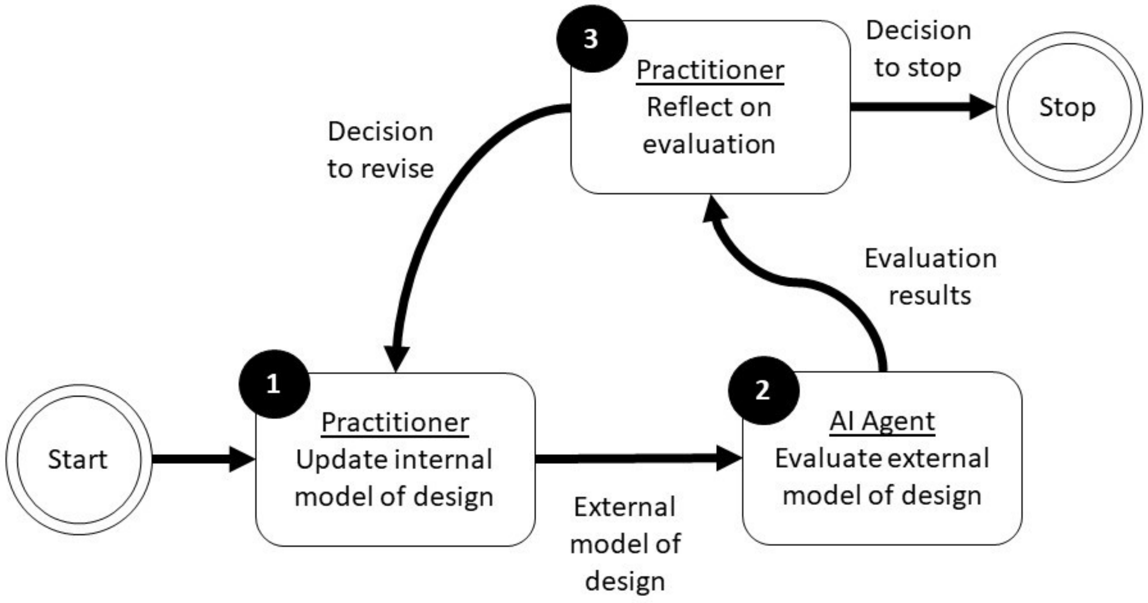


Figure 1.1: Hypothesized process of incremental design revision supported by an AI agent

3. **What form should the practitioner's externalized design model take to be given as input to the AI agent?** Answering this question is necessary to develop the AI agent in detail.

All three of these questions relate to each other. Question 1 sets up the goal for the agent in question 2 to support, and question 3 describes what knowledge the agent will have with which to work. Thinking in the reverse direction as well, the answer to question 1 must, practically-speaking, be addressable through computational support, and the knowledge from question 3 needs to facilitate the reasoning strategies taken by the AI agent. As such, one must answer these three questions simultaneously to achieve the desired harmony among them.

In this dissertation, a design version is better than another version if it has superior internal and external consistency. Internal consistency is meant as the extent to which aspects of the model agree with each other. External consistency is meant as the extent to which the model accurately represents (relative to the purpose of the model) the modeled concept as it is in the world. For candidate designs, which have no real world counterpart, external consistency instead is meant as the extent to which the model aligns with those of the practitioner's teammates or any external representation of the design (assuming either exist). In summary, internal consistency is an inward-looking notion of alignment, whereas external consistency is outward-looking.

Given this answer, an AI agent can support incremental design revision by checking for and reporting on potential issues related to internal and external consistency for a given design. To do so, the agent reasons about the practitioner's design model, which is assumed to be articulated in some externalized form to be fed as input to the AI agent. What might that form take? Prior work advocates for function to be addressed during the conceptual design stage, so this dissertation builds on this and operationalizes the externalized form as a functional model. In particular, this dissertation assumes that the external model will describe a function to structure mapping with behaviors, or process descriptions, used to explain how the structures achieve the functions.

Given this knowledge representation, the question of how the AI agent will check for internal and external consistency issues can be answered concretely. Regarding internal consistency, the agent will check that the functions are achieved by the structures through the behaviors as claimed. Assume that the behaviors are themselves explicitly stated (as opposed to being implied through structural relations). The agent will also check that these behaviors proceed as claimed. The agent will do all of this through simulating, or executing, the behaviors in the input model and leveraging those results to check the function and behavioral claims.

Simulation is used to computationally check for internal consistency. In a general sense, checking for internal consistency is a hard problem because one needs to determine how to automatically reason across complex models to identify internal consistency issues. It is also a design challenge that involves considering the relationship between the reasoning strategy and the representation over which it operates. In a specific sense, simulation itself brings related challenges. For example, qualitative reasoning is a desirable style of reasoning in the conceptual design stage due to not requiring precise quantitative knowledge. How should this reasoning be represented in the modeling language? How should the simulator process qualitative equations? These technical questions needed to be answered to enable internal consistency checking through simulation, and the relationship between knowledge representation and reasoning strategies shown by this example is emblematic of the challenge in internal consistency checking.

Regarding external consistency, the agent will check that the functions, structures, and behaviors of the input model accurately reflect the modeled concept or, for the candidate design, the modeled concept as envisioned. It will do so by analogically comparing the input model against an alternative model of the same concept. Where does the alternative model come from? It could have been automatically generated from a trusted source like an academic paper, or it might come from a teammate if in the context of collaborative design. In the case of interdisciplinary teams, a teammate may have a greater expertise on the topic than the practitioner, thus allowing the teammate's model to act as an authoritative source. Although it does not address the authority of a teammate's internal understanding, [64]'s example of an instance of cognitive dissonance between design team members in an education context demonstrates the plausibility that two members on a BID team may hold different internal understandings of knowledge in this context. The alternative model need not be considered anything as authoritative as ground truth. In all circumstances (regardless of the authority of the alternative model), identified differences act as triggers for further investigation because they may represent misconceptions or incomplete knowledge in the practitioner and/or the author of the alternative model.

Computationally checking for external consistency is a hard problem because one does not know how or to what extent the input model to the AI agent will differ from the modeled concept (or envisioning). Narrowing the problem to model-to-model comparison simplifies this to the ontology of the modeling language. That is, one can now enumerate the finite ways in which differences might arise (e.g., different components or different state conditions). However, it remains unknown which of these differences will manifest and to what extent. This is a challenge because difference detection should only identify valid (i.e., salient and actual) differences, for producing false positives reduces the efficacy of the approach since practitioners must then sift through potentially erroneous results.

Finally, in light of the above discussion, one can refine and restate the role of biologically inspired design (BID) in this dissertation. One, BID serves as a real-world, socially meaningful context for research into design and analogy. Two, BID motivates the content of knowledge: biological and technological (engineering) systems. Three, BID as design practice (and a focus on conceptual design as the specific phase of this practice) motivates the representation of knowledge: functional models. Four, that BID engages interdisciplinary knowledge motivates checking the accuracy of practitioner's knowledge since it is possible that practitioners are not experts in biology or engineering. Note that regarding this point: although the DESC AI agent focuses on model evaluation, these models are intended to be representations of a practitioner's internal understanding of the concepts. Thus, DESC is *really* evaluating the practitioner's knowledge, not just evaluating models. Five, lastly and related to the previous point, if BID occurs in a collaborative design context, the potential for teammates to exist (which is admittedly possible in any design context) and to have different expertise as the point-of-view practitioner (e.g., it is possible that the practitioner is an engineer and a teammate is a biologist, which is related to BID's interdiscplinary nature) affords alternative models on the same topic that are potentially more authoritative than the practitioner's, which in turn improves the plausibility of DESC's Comparison technique.

1.1 Thesis Statement

The preceding discussion builds towards the thesis statement of this dissertation, which is as follows:

> The conceptual phase of biologically inspired design entails incremental design revisions across multiple knowledge entities. An AI agent may support systematic and repeatable incremental design revision by checking the internal and external consistency of functional models of candidate designs as well as biological source cases and prior designs.

1.2 Research Questions and Hypotheses

The research questions and hypotheses of this dissertation are formalized, concise versions of the discussion at the beginning of this chapter. One can also think of these questions as themselves design questions since they relate to the development of an AI agent.

The first research question represents the high-level goal of this work. It is the same as the second question in the beginning part of this chapter.

- **Research Question 1.** How might an AI agent support systematic and repeatable incremental design revision?

  Hypothesis 1. It may do so through checking the internal and external consistency of design models.

The second research question drills down into the first half of Hypothesis 1, relating to internal consistency checking. Its hypothesis is broken down into two parts. The first part, 2.1, refers to the process by which the AI agent will check for internal consistency. The second part, 2.2, refers to the representation of the content over which the agent will operate to do so.

- **Research Question 2.** How might an AI Agent check a design model for internal consistency?

Hypothesis 2.1. It may do so by simulating the model and using the results to check the process and functional claims made by the model. The Simulation technique of DESC illustrates this hypothesis.

Hypothesis 2.2. The SBF* functional modeling language provides the syntax and semantics needed for automated simulation.

The third research question connects with the second half of Hypothesis 1, relating to external consistency checking. Since the method of comparison does not require any particular knowledge content (although it does assume structured knowledge as with the rest of this dissertation), only the process part of the answer is given an explicit hypothesis.

- **Research Question 3.** How might an AI agent check a design model for external consistency?

  Hypothesis 3. It may do so by analogically comparing a given model against another model of the same design, where differences represent potential issues. The Comparison technique of DESC illustrates this hypothesis.

Hypothesis 3 raises the question of how to map two models, which is given an explicit research question in number 4, below. The hypothesis to this research question proposes a novel analogical mapping technique. Note that another analogical mapping technique could also satisfy this research question, so implicit in Hypothesis 4 is that this is an exploratory attempt to develop a better technique than a method without its ideas.

- **Research Question 4.** Analogical comparison implies mapping. How might an AI agent map two models?

  Hypothesis 4. It may do so by decomposing the mapping task, solving the subtasks, and using solutions from earlier subtasks to constrain and inform solutions to future subtasks. Compositional Mapping illustrates this hypothesis.

1.3 Supporting Incremental Design Revision with an AI Agent

The core contribution of this dissertation is the development of an AI agent that will take an external model of a design, evaluate it in terms of its internal and external consistency, and produce feedback for the practitioner in the service of supporting incremental design revision. This is represented as step 2 in Figure 1.1. It is shown in slightly more detail as an isolated, high-level process in Figure 1.2.

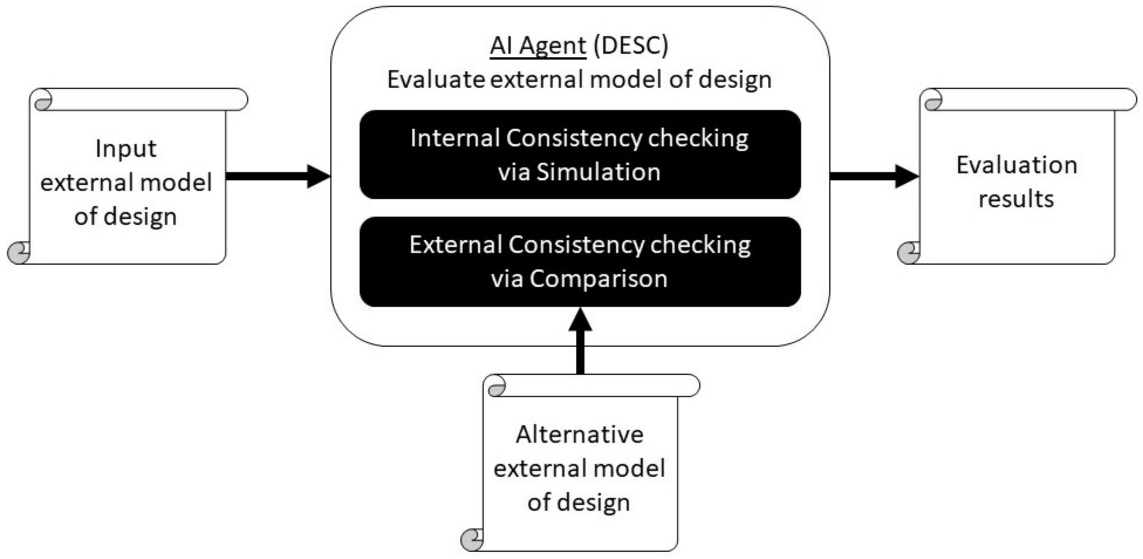

AI Agent (DESC)
Evaluate external model of design
Internal Consistency checking via Simulation
External Consistency checking via Comparison
Input external model of design
Evaluation results
Alternative external model of design

Figure 1.2: High-level process of the Design Evaluation through Simulation and Comparison (DESC) AI agent

Doing so raises three questions. First, how should the input design model be represented and (zooming ahead for a moment) how should the alternative design model used for Comparison be represented? Second, how should the AI agent evaluate or check the input model for internal and external consistency issues? Third, what should the outputted results look like? This dissertation focuses on the first two questions and leaves refining an answer to the third question for future work. The next sections will briefly discuss how this dissertation addresses the first two questions after first developing a running example.

1.3.1 Running Example

It is helpful to ground the following discussion in a concrete example so that not all topics need to be exclusively discussed in the abstract. Thus, consider a facet of the biologically inspired design case described by McKeag [100]. Here, Shinkansen Train designers/engineers were tasked with developing a faster train to meet a specified goal: "get a passenger from Shin-Osaka to Hakata station at Fukuoka in under 2 hours and 20 minutes" [100, p. 18]. This would require their train to reach near 350kph. They were able to build a fast enough train, but their new train grew in noise with its speed, and they needed to be considerate of noise standards. Reducing the train's noise thus became a design problem.

One way in which the train produced noise was through an aerodynamic phenomenon. The train's pantographs (linkages at the top of the train) created vortices in air (turbulence) that in turn created sound. Figure 1.3 depicts an example of a train pantograph. (This example is actually the completed "Winggraph" that was developed as the design solution for the case currently being discussed.)

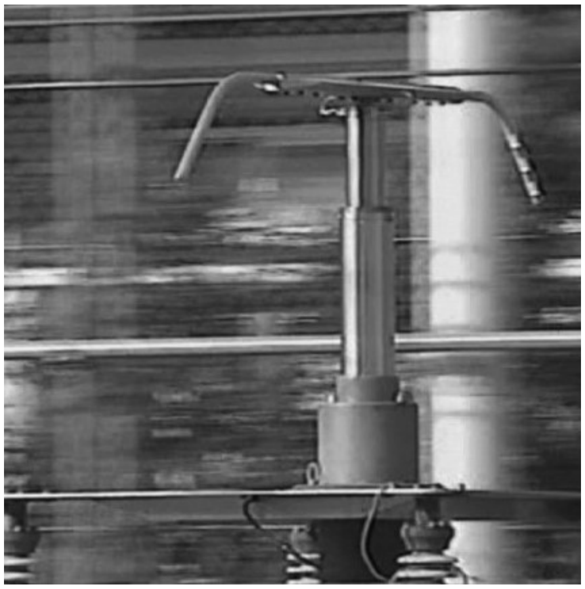


Figure 1.3: An example of a train's pantograph: the completed "Winggraph" design solution for the Shinkansen Train example case. Adapted from [100]

In addition to applying non-analogical engineering practices, designers resolved this problem by inspiration from the way owl wings help an owl reduce sound while flying. The fimbriae, a set of serrations on the leading edge of owl wings, generate small vortices in the air that produce less sound than larger vortices. Figure 1.4 depicts these serrations and shows how air flows over them.

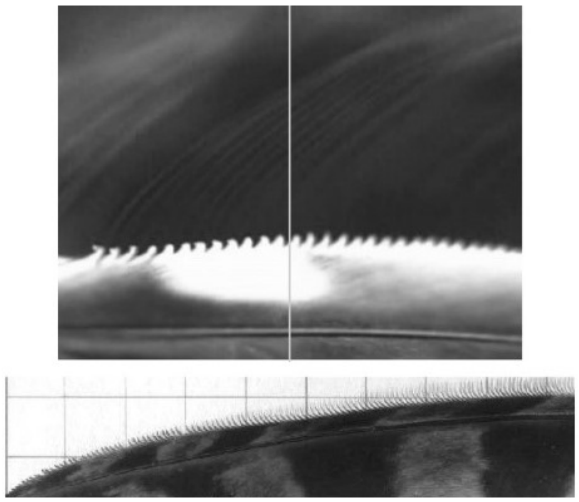


Figure 1.4: Serrations (consisting of fimbriae) on owl feathers, including (in top picture) air flow across the feather. Images are adapted from [100]

In turn, the Shinkansen Train designers developed a small vortex generator (structures applied to the redesigned pantographs) to create the same effect and help resolve this design challenge. Figure 1.5 shows this small vortex generator (the small triangular pieces). This plus other modifications led to the "Winggraph" depicted in Figure 1.3, which helped resolve the noise problem for the train.

Figure 1.6 depicts this case in terms applicable to DESC, with pointers to the two models (using the code names given in Appendix D) developed for this dissertation that address the associated knowledge entities. Although the design process ran to completion and thus extended past the conceptual design stage (and it is unclear at what stage the various knowledge entities appeared), this dissertation frames it in the context of conceptual design as if it hypothetically existed in that stage for the sake of an example. The noisy Shinkansen train acts as the prior design. A correct and complete understanding of how this train

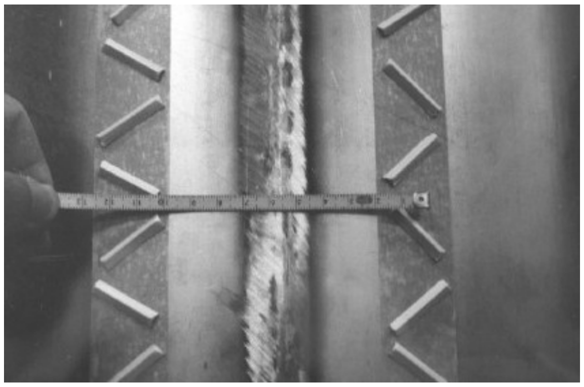


Figure 1.5: Small vortex generator (the triangles) that was developed to partially address the aerodynamic noise issue. Adapted from [100]

works (especially as it relates to creating noise) is important to the design process, for the designers need to modify the train in the proper ways to actually solve their problem. Specific to the situation described above, the designers need to know how the train creates aerodynamic sound and (eventually) the role that the train's pantographs play in that process. A misunderstanding here might cause designers to make changes to parts of the train that are actually irrelevant (or less important) to aerodynamic sound creation.

The owl wing (or, rather, the way in which the owl wing produces low sound for the in-flight owl) is the source case used by the design process. A correct and complete understanding of this system is also important because it will be used to infer a solution. The Owl Wing model developed for this dissertation is representative of a simple version of this topic.

Finally, the redesigned train (or, rather, a hypothetical version of this in the conceptual design phase) is the candidate design. It is important to evaluate this design to ensure that the problem is actually solved as expected. The Train model developed for this dissertation is representative of a simple version of this topic.

Prior Design: The noisy Shinkansen train. (Not modeled by this dissertation.)

Source Case: The way the owl wing enables low-sound during flight. Modeled in this dissertation by the Owl Wing model.

Candidate Design: The (hypothetical) conceptual design phase version of the modified train with a small vortex generator. Modeled in this dissertation by the Train model.

Figure 1.6: Shinkansen train case of biologically inspired design, derived from [100], broken into and characterized as knowledge entities relevant to DESC

1.3.2 Structure-Behavior-Function Star (SBF*)

The first question in the high-level design evaluation process is: how should the input design model be represented and how should the alternative design model used for Comparison be represented? In response, this dissertation assumes that designs (which includes source cases, prior designs, and candidate designs) are articulated in a version of the Structure-Behavior-Function (SBF) modeling language that is called Structure-Behavior-Function Star (SBF*). SBF [65] is a structured, functional model representation. It represents a concept in terms of its physical structure, its behaviors (or mechanisms), and its functions, which are its designed or (especially when considering natural systems) perceived purposes.

Why use SBF*? First, it is a functional model, and these are appropriate in the conceptual design stage because according to [112] this stage involves working with functions and thus a modeling language that captures functions aligns with this thinking. Second, it is a member of the SBF family of languages, and prior research [162] has demonstrated that versions of the SBF language can capture biological and technological systems, can be automatically retrieved in the BID context [160, 172], and can be used in automated analogical design [66] of technological systems, so such a modeling language is plausible for articulating the concepts in the biologically inspired design context and for reasoning over them by a computer. Third, SBF* as a specific language that aligns with this dissertation is appropriate because SBF* adds syntax and semantics to enable automated simulation, making it appropriate for the kind of reasoning done by the DESC AI agent for its internal consistency evaluation technique.

An SBF* model has the following parts. The structure submodel of an SBF* model describes a design's components, their attributes, and any connections between them. Figure 1.7 depicts two snippets from the structure submodel of the aforementioned Owl Wing model. The Owl Wing model contains the component `Owl` with two attributes, `Moving` and `NoiseCreated`. (Connecting points are ignored in this dissertation for simplicity.) The model also contains a connection between the owl's `Wing` and the main `Owl` structure called `WingIsPartOfAnOwl`.

```
Owl is a component
  It has a connecting point named C1
  It has an attribute Moving of type Qualitative, described as
    "False, True"
  It has an attribute NoiseCreated of type Qualitative,
    described as "Zero, Very_Low, Low, Medium, High"

Wing.C1 is PartOf Owl.C1 is a connection named
  WingIsPartOfAnOwl
```

Figure 1.7: Snippets from the structure submodel of the Owl Wing model

The functions of a design describe its purposes or goals, either those intended by the designer or (especially in case of a biological system) those that are ascribed to it. A function optionally declares a representative verb to describe it, and it contains a set of conditions that it guarantees will be true in the world at its completion. These are called the function's provides conditions. Each function also points to a behavior that achieves it. Figure 1.8 depicts the function `OwlFliesSilently` from the Owl Wing model. This function ascribes flying silently as the purpose of the owl. The function's provides condition says that the amount of `NoiseCreated` by the `Owl` will be `Very_Low`, fitting with the notion of flying silently. (Admittedly, that it produces some noise is somewhat dissonant with the function name.)

```
OwlFliesSilently is a function
  It requires nothing
  It provides: (
    Owl.NoiseCreated = "Very_Low"
  )
  It is achieved by the behavior OwlFliesSilentlyBehavior
```

Figure 1.8: Function from the Owl Wing model

Zooming out of this single function, Figure 1.9 diagrammatically presents the functional decomposition of the Owl Wing model. White boxes represent the functions; each box declares the function's name and set of provides conditions. Black rounded boxes represent behaviors, which achieve functions. Arrows are drawn from functions to their associated behaviors and from behaviors to functions to represent function-subfunction relationships with mediating behaviors. One reads this diagram from the top to bottom, so `OwlFliesSilently` is the topmost function.

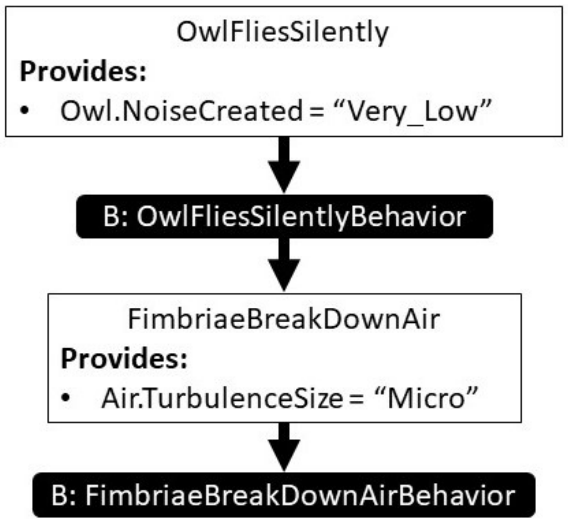

OwlFliesSilently
Provides:
• Owl.NoiseCreated = "Very_Low"
B: OwlFliesSilentlyBehavior
FimbriaeBreakDownAir
Provides:
• Air.TurbulenceSize = "Micro"
B: FimbriaeBreakDownAirBehavior

Figure 1.9: Functional decomposition for the Owl Wing model

The behaviors describe the processes or mechanisms the system undergoes. Each behavior provides a causal description in the form of a state-transition diagram. Each state defines a set of conditions in the world that are true at a moment in time. Transitions between states are annotated with explanations for why and how the system moves from one state to the next, sometimes describing how attribute values change between states. Among other things, an explanation can point to a function for why its transition happens. Since all functions point to an achieving behavior and since a behavior can point to one or more functions in its transition annotations, such relationships form a function-subfunction decomposition of the design.

Figure 1.10 depicts a snippet from the Owl Wing model's `OwlFliesSilentlyBehavior` behavior, which is the behavior that achieves the aforementioned `OwlFliesSilently` function. This snippet describes the owl initially still in the state `StartState` then deciding to move, which causes air to move past it (transition `T1`), resulting in state `State2`.

1.3.3 AI Agent: Design Evaluation through Simulation and Comparison (DESC)

The second question in the high-level process of design evaluation is: how should the AI agent evaluate or check the input model for internal and external consistency issues? In response, this dissertation presents an AI agent to evaluate functional models of designs called Design Evaluation through Simulation and Comparison (DESC). DESC encapsulates Hypotheses 2.1 (and 2.2 by virtue of it motivating and leveraging features of SBF*) and 3.

As suggested by the name, DESC contains two parts: Simulation and Comparison. These two independent techniques provide complementary means to evaluate a functional model of a design. The first evaluates the internal consistency of a model by detecting modeling issues through simulation. The second evaluates the external consistency of a model by comparing it against another model (specifically another SBF* model) of the same topic. Detailed descriptions of these techniques are given in Chapter 4.

```
OwlFliesSilentlyBehavior is a behavior

  StartState is the start state
    It has the condition: (
      Owl.Moving = "False"
      and Air.MovingPastOwl = "False"
      and Owl.NoiseCreated = "Zero"
    )

  T1 is a transition from StartState to State2
    It has the explanation: (
      the external stimulus named OwlDecidesToFly, described as
          "eq: Owl.Moving is equal to Owl.Moving.True"
      and the equation AirMovesPastOwlEQ, described as "eq: Air
         .MovingPastOwl is directly proportional to Owl.Moving:
         After - Owl.Moving:Before"
    )

  State2 is a state
    It has the condition: (
      Owl.Moving = "True"
      and Air.MovingPastOwl = "True"
    )
```

Figure 1.10: Snippet of a behavior from the Owl Wing model

1.3.3.1 Simulation Technique of DESC

The Simulation technique of DESC embodies Hypothesis 2.1 and, through using the modeling features, addresses 2.2. Assume that a practitioner articulates their understanding of a design through a functional model that contains descriptions of the functions of the design along with the processes (behaviors) that achieve those functions. Hypothesis 2.1 proposes that a computer can by check for the internal consistency of a model by simulating the model and using the results to check the process and functional claims made by the model. This in turn necessitates the hypothesis that the modeling language used has syntax and semantics for automated simulation (Hypothesis 2.2), which is achieved by using SBF* models.

Figure 1.11 depicts the high-level process of Simulation. Simulation accepts an input

SBF* model in step 1. As the output of this step, Simulation produces an inferred behavior for each input behavior, which differs in that all states except for those that begin the behaviors have their inputted conditions replaced by conditions inferred during simulation. In step 2, Simulation compares the states in the inferred behaviors against the input SBF* model's behaviors and determines any inconsistencies between the two, which are reported in the results of Simulation. In step 3, Simulation compares the output of the inferred behaviors against the functions in the input SBF* model to determine the extent to which those behaviors produce output consistent with the functions. Issues here are also added to the results. In the operational implementation of Simulation created for this dissertation, the final form of these results take the form of, essentially, a human-readable list of issues.

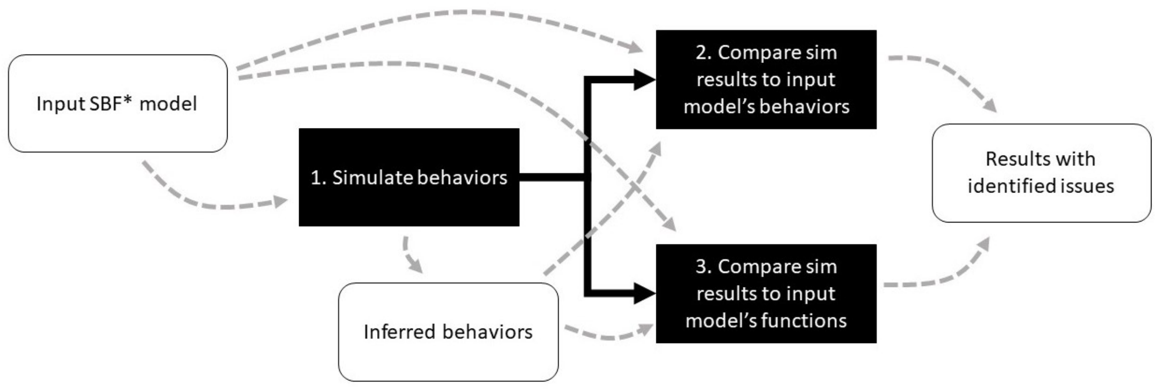

Input SBF* model
1. Simulate behaviors
Inferred behaviors
2. Compare sim results to input model's behaviors
3. Compare sim results to input model's functions
Results with identified issues

Figure 1.11: Overview of DESC's Simulation technique

Figure 1.12 depicts an example snippet of Simulation's output that contains its feedback. This output has been formatted for this document but is similar in appearance to the literal output. These results were created during one of the Computational Experimentation trials (see below), involving a version of the Owl Wing model with an intentionally incorrect equation in one of its behaviors and involving the Simulation technique with all of its features enabled[2]

[2]To be precise in terms of that experiment, this is trial 2 with the Complete configuration of Simulation and the Wrong Equation configuration of the Owl Wing model.

I ran a simulation to verify your functions and behaviors. Below, I point out any issues that I identified with them.
Behaviors

- For OwlFliesSilentlyBehavior:
  - For state State2:
    - You said Air.MovingPastOwl = True. I think that it should be False.

Figure 1.12: Feedback part of Simulation's output for an ablated version of the Owl Wing model

1.3.3.2 Comparison Technique of DESC

The Comparison technique of DESC embodies Hypothesis 3. Assume that the practitioner articulates their understanding of a design through a structured representation. Specifically, this dissertation assumes a SBF* model. Hypothesis 3 proposes that a computer can evaluate the external consistency of this model by analogically comparing a it against another model of the same design, where differences represent potential issues.

The other, alternative model could come from a design teammate, a knowledgebase, or an oracle. Note that the other model does not need to be considered ground truth, so to speak, for the Comparison technique. If the alternative model is more likely to be correct (e.g., consider: the topic is a biological system, the practitioner is an engineer, and the alternative model comes from a biologist teammate) then raised differences are likely problems with the practitioner's model. If the alternative model is less likely to be correct, (e.g., consider the previous situation but the expertise roles were reversed) then raised differences are likely problems with the alternative model. It could also be the case that neither model should be assumed more correct than the other (e.g., perhaps both teammates are novices in the domain).

In all of these cases, the appropriate response to identified differences is to use them as triggers for further investigation. They highlight areas of concern that may represent misconceptions or incomplete knowledge, and since neither model is viewed as ground truth, the problem may lie in either (or both!) models. Differences may also reflect different modeling decisions and thus not necessarily require reconciliation. Comparison's utility is in raising possibilities to be deliberately addressed by the practitioner and any involved teammates.

Figure 1.13 depicts the overall process of Comparison. An Input SBF* model and an Alternative SBF* model are accepted as input. In step 1, Comparison aligns the models (a.k.a., maps them) using some mapping algorithm. In step 2, it leverages that mapping to detect differences between them.

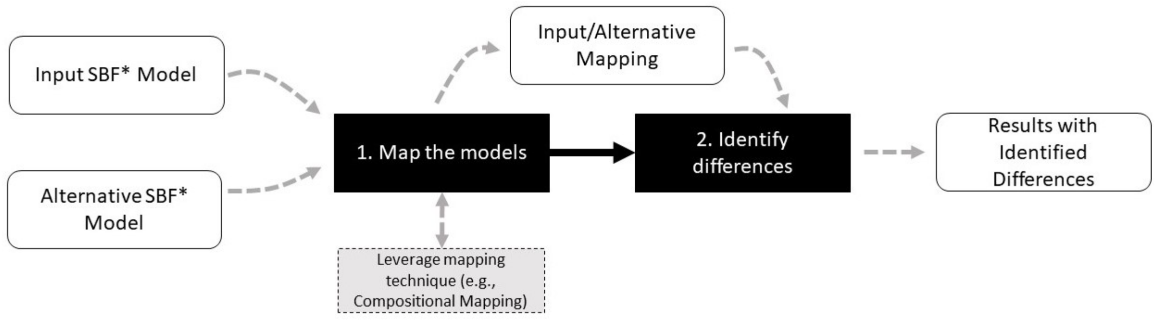

Input SBF* Model
Alternative SBF* Model
Input/Alternative Mapping
1. Map the models
2. Identify differences
Results with Identified Differences
Leverage mapping technique (e.g., Compositional Mapping)

Figure 1.13: Overview of DESC's Comparison technique

With this process comes another question: how should one align (or map) two functional models to facilitate difference detection? Hypothesis 4 proposes that an AI agent may do mapping by decomposing the mapping task, solving the subtasks, and using solutions from earlier subtasks to constrain and inform solutions to future subtasks, which is embodied by the novel mapping technique Compositional Mapping.

Compositional Mapping decomposes the alignment task into a series of subproblems, each corresponding to a logical partition of the knowledge representation. For example, in SBF*, the partitions created for this dissertation were the structure submodel, the behaviors (as a submodel), a behavior, a state, a transition, the functions (as a submodel), and a function. Compositional Mapping solves the subproblems in a defined sequence, propagating prior subproblem solutions as constraints to guide and constrain future subproblem solving.

Implicit in Hypothesis 4 is the belief that it will be superior to a technique missing these ideas, which is represented in this dissertation by a technique called Uniform Mapping that leverages prior work to apply a more conventional, Structure-Mapping Engine [44] approach to mapping two SBF* models.

Figures 1.14 and 1.15 depict example snippets of Comparison's outputs that show its feedback. This output has been formatted for the document but is similar in appearance to the literal output. These results were created during one of the Computational Experimentation trials (see below), involving the Train model, where one of the two Train models used for comparison had one of its subfunctions and associated subbehaviors integrated into their superfunction and superbehavior, respectively and where the Comparison technique was using Compositional Mapping[3]. The Train model–which models the candidate design of the running example–is used here for an example instead of the Owl Wing model because the Owl Wing model's Comparison results for the Computational Experimentation all found no differences.

1.3.4 Situating DESC in Processes

Design Evaluation through Simulation and Comparison (DESC) is an AI agent to aid design evaluation, intended to support an incremental design revising process. The prior sections describe how DESC works at a high-level, mostly decontextualized from broader situations. This section addresses that gap by speculating where one might situate DESC within a generic conceptual design process, within two biologically inspired design processes, and within an abstract analogical reasoning process.

1.3.4.1 Within Conceptual Design

Pahl & Beitz [112] describe a multi-stage, generic design process. The main phases of this process along with their inputs and outputs are depicted in Figure 1.16. This dissertation

[3]To be precise in terms of that experiment, this is trial 5 with the Compositional Mapping (with interaction) configuration of Comparison and the Behavior/Function Reorganization configuration of the Train model.

I compared your model against one from my knowledgebase. Below, I point out differences between the two of them. When looking to make changes to your model based on this information, please note that my model may not necessarily be correct.

Differences in Behaviors

- I matched your behavior TrainGeneratesAerodynamicNoiseBehavior to my behavior TrainGeneratesAerodynamicNoiseBehavior.
  - I matched your state StartState to my state StartState.
    - I did not find a match for your condition Train.EngineThrottle = Off.
  - I did not find a match for your state Sub_TrainThrottleIsOn.
  - I did not find a match for your transition Sub_T1.
  - I matched your transition Sub_T2 to my transition T1.
    - I did not find a match for your explanation E1 of type Equation with description "eq: Train.Accelerating is directly proportional to Train.EngineThrottle:Before".
    - I did not find a match for your explanation E2 of type Equation with description "eq: Train.Velocity is directly proportional to Train.Accelerating:After".
    - I did not find a match for my explanation EngineCausesTrainToAccelerate of type Function.
- I did not find a match for my behavior EngineCausesTrainToAccelerateBehavior.

Figure 1.14: Feedback part of Comparison's output (part 1) for a modified version of the Train model

scopes its work on the conceptual design phase of this process, annotated in the figure. In conceptual design, one develops the functions required to solve the design problem, develops an initial structure of the design solution, and outputs a candidate design. How would DESC be situated within the conceptual design process?

Figure 1.17 depicts Pahl & Peitz's [112] conceptual design process, annotated with where DESC might apply. This boils down to two fundamental spaces. First, DESC could be used to evaluate any retrieved existing solutions (prior designs) during the phases

Differences in Functions

- I matched your function TrainGeneratesAerodynamicNoise to my function TrainGeneratesAerodynamicNoise.
  - I did not find a match for your condition Train.Accelerating = On.
- I did not find a match for my function EngineCausesTrainToAccelerate.

Figure 1.15: Feedback part of Comparison's output (part 2) for a modified version of the Train model

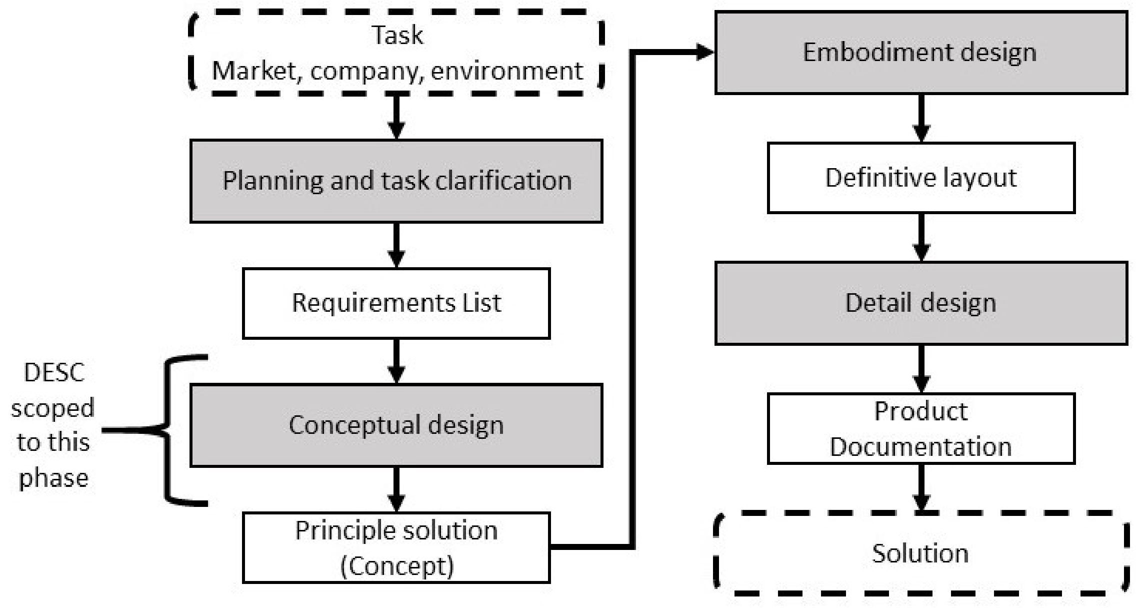


Figure 1.16: Generic design process annotated with the scope of this dissertation. Contents in this figure are derived from [112, p. 130]. Shaded boxes represent main phases in the process. Smaller unshaded boxes represent input/output. Dashed boxes reflect the input and output of the total design process

where one is developing the function structures, retrieving these solutions, and developing the design's structures. Both Simulation and Comparison would be equally useful here. The practitioner could use DESC to incrementally refine their knowledge of these prior designs. Second, DESC could be used to evaluate the candidate design (what [112] terms the "principle solution") before moving on to the next stage in the design process, preventing wasted work due to repairs or backtracking. Again, this could be part of an incremental or formative process, supporting the refinement of the candidate design over time.

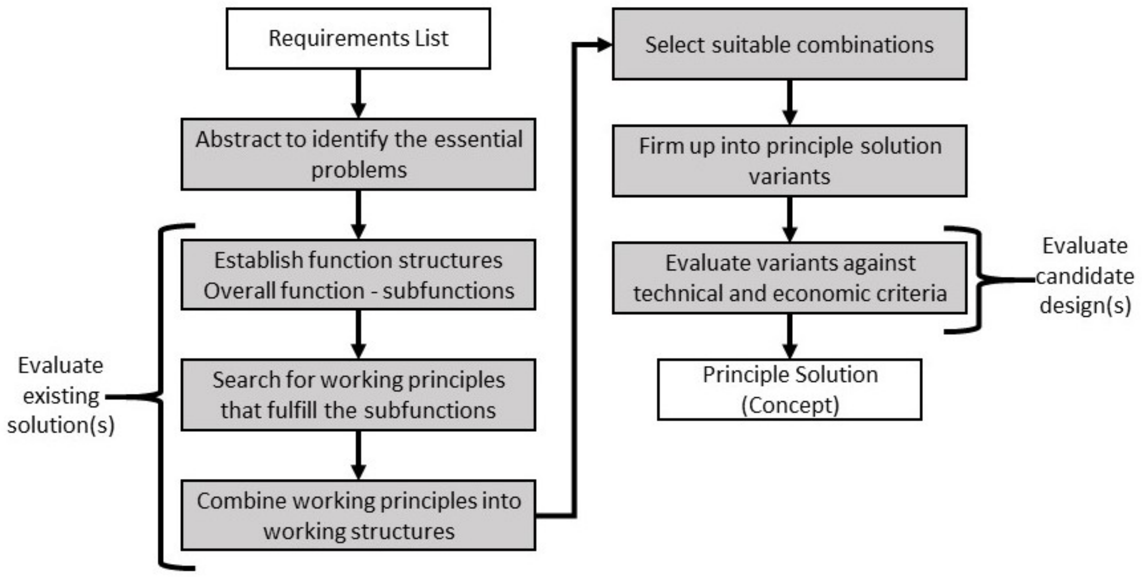

Requirements List
Abstract to identify the essential problems
Establish function structures Overall function - subfunctions
Search for working principles that fulfill the subfunctions
Combine working principles into working structures
Evaluate existing solution(s)
Select suitable combinations
Firm up into principle solution variants
Evaluate variants against technical and economic criteria
Evaluate candidate design(s)
Principle Solution (Concept)

Figure 1.17: Generic conceptual design process annotated with where to apply DESC. Contents in this figure are derived from [112, p. 160]. Shaded boxes represent process steps. Smaller unshaded boxes represent input/output

1.3.4.2 Within Biologically Inspired Design

The previous section situated DESC in an abstract conceptual design process. This section switches to the more concrete biologically inspired design process. Vattam et al. [161] discuss processes and tasks that student designers underwent in a biologically inspired design course. Where might DESC be situated, at a high level, in the problem-driven and solution-driven processes mentioned in this article?

To paraphrase their description of the problem-driven process and group it into three phases (it is normally six): (1) identify, define, and biologize a problem; (2) retrieve and define biological solutions; and (3) extract and apply abstracted mechanisms to define a trial design solution. If a deficient technological solution is part of the problem defined in part (1), one could use DESC during this phase to evaluate this aspect of the problem and incrementally refine knowledge of that solution. This could reduce the chance of issues that might impact later phases, such as retrieving biological solutions or defining a trial design solution. Phase (2) also presents an opportunity to use DESC, for one could use the techniques to evaluate one's understanding of the retrieved and defined biological solutions and to incrementally refine that understanding. Issues arising from this application could lead to changes that prevent issues caused by transferring misconceptions into the trial design, or they could even lead to rejecting a retrieved biological solution if the changes lead to realizing the biological solution is not as apt to the design problem as once believed. Finally, DESC could also be used in phase (3) to check the trial design. Again, this could be also applied in an incremental fashion rather than viewing it as a one-shot activity. If this is a truly novel design, it is unlikely that an alternative model would exist for comparison, but (i) Simulation could still be ran to evaluate the trial design and (ii) design teammates could each construct their own models of the trial design, enabling Comparison to reveal any dissonance amongst how team members conceptualize the design.

Next is the solution-driven process. The authors provide a simple description of the process: "In contrast, in the solution-driven approach... designers began with a biological source of interest. They understood (or researched) this source to a sufficient depth to support extraction of deep principles from the source. This was followed by finding human problems to which the principle could be applied. Finally, they applied the principle to find a design solution to the identified problem" [161, pp. 469-470]. At an abstract level, this roughly corresponds to stages (2), (1), and then (3) of the above three-phase version of their problem-driven process ([161]'s version of the solution-driven process is itself seven phases), except that designers begin with a biological solution and retrieve a relevant design problem.

DESC could still apply to all three phases of the process. Regarding the first phase, one should still evaluate the biological solution and refine one's knowledge, although without notions of aptness. Developing one's understanding of the biological solution is perhaps more important in this process because issues could effect what design problems get retrieved in addition to impacting the design solution. Regarding the second phase, one should still evaluate and refine the part of the design problem that contains a deficient technological solution (if it contains such a thing), for how one understands the design problem impacts how one will try to solve it. Here, like in the previous process' second phase, revisions to one's understanding of the design problem may lead to rethinking its aptness. Finally, DESC in the last phase applies the same as in the problem-driven process.

This quick analysis of where to situate DESC within two biologically inspired design processes illustrates that design evaluation (and the associated incremental revising it affords) may play a valuable role throughout biologically inspired design.

1.3.4.3 Within Analogical Reasoning

Biologically inspired design is an instance of analogical design, which includes analogical reasoning. Zooming out to an abstract analogical reasoning process, where would DESC apply? Consider a hypothetical four-stage analogical reasoning process containing the stages (1) retrieval, (2) mapping, (3) transfer, and (4) evaluation. This process, depicted in Figure 1.18, is more or less a hybrid of those described by [43] and [76]. The process starts by assuming that the reasoner (i.e., the agent conducting the analogical reasoning episode) has a Target concept in mind, which is that concept about which the reasoner wants to infer knowledge. In the retrieval stage, the reasoner finds a Source concept from which to transfer knowledge for inferencing. In the mapping stage, the retrieved Source concept is mapped to (a.k.a., aligned with) the Target concept. This matches elements in the Target concept to the Source concept. In the transfer step, the generated mapping is leveraged to transfer new information into the Target concept. This is analogical inferencing. Finally, in the evaluation stage, the modified Target concept is evaluated.

One might situate DESC at multiple places in this process, represented by the lettered circles in Figure 1.18. First, for the sake of argument, make the simplifying assumption

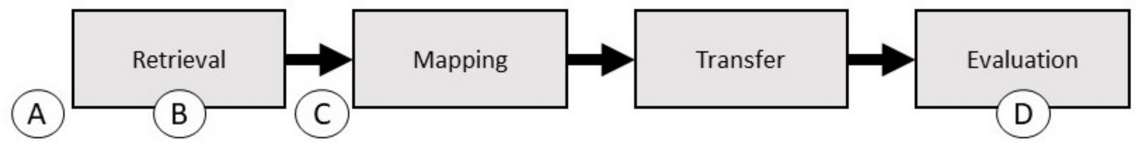


Figure 1.18: A 4-Stage analogical reasoning process. Shaded boxes represent process stages. Circles represent where one could apply DESC

here that all concepts are represented as SBF* models. One could use DESC before the retrieval stage begins, represented by circle A in the figure. Here, one should evaluate the Target concept and all potential Source concepts, and this can be done in an incremental manner until the reasoner is satisfied by their understanding of all the concepts that will potentially be used. Doing this before retrieval is important because any issues in the Target and Source concepts could impact retrieval, resulting in a suboptimal Source concept being retrieved. It is also interesting to note that evaluating the Target concept at this stage may eliminate the need for analogical reasoning if an appropriate Alternative concept exists, for perhaps the reasoner could garner all the needed new knowledge about the Target concept from comparison alone[4].

One could also use DESC during the retrieval stage. This is represented by circle B in the figure. Here, one should evaluate Source concepts as they are retrieved. In doing so, the reasoner could both repair Source concepts as needed and also develop a notion of confidence in each Source concept relative to the issues identified and the amount of incremental revising the reasoner wishes (or has the time and energy) to do. This could be used as a metric to help decide which Source concept to take forward into the next stage of reasoning.

One could use DESC after the retrieval stage but before the mapping stage, represented by circle C in the figure. Here, one should evaluate both the Target and Source concepts, repairing them (perhaps incrementally) as needed. This is important because uncaught

[4]Although this is perhaps a circular argument given that the Comparison technique involves analogical mapping.

issues could impact the mapping and transfer stages of analogical reasoning, potentially leading to an erroneous modified Target concept as output.

Finally, one could use DESC during the evaluation stage. During this stage, DESC could be used to evaluate the modified Target concept (something unavailable at the previous locations), potentially leading to the reasoner repairing that concept (perhaps incrementally) or conducting additional iterations of one or more steps of the analogical reasoning process. If Source and Target concepts were not yet evaluated, the reasoner could also evaluate them at this point. If issues are discovered, the reasoner could conduct incremental revising of those concepts then step back to repeat the mapping and transfer stages. Alternatively, discovered issues could simply alert the reasoner that the modified Target concept may be erroneous, so the reasoner should go forward with a skeptical eye when using it in the future.

1.4 Assumptions & Limitations

This section describes the top-level assumptions made by this dissertation and a set of the dissertation's limitations. The set of assumptions are as follows:

- **Assumption:** Each design is articulated as a syntactically well-formed (although may contain semantic errors) Structure-Behavior-Function Star (SBF*) model. SBF* models are sophisticated structured knowledge representations that require some degree of familiarity and require time to construct. That said, as functional representations they are within the domain of plausibility for the conceptual design stage targeted by this dissertation.

- **Assumption:** The Simulation technique of DESC assumes the input SBF* model has behaviors, simulation-capable explanations, that there are no loops in the function decomposition, and that the modeler articulated their behavior state conditions along the lines assumed by implicit value forwarding (i.e., missing attribute values mean unchanging values). All of these assumptions hold in the six models constructed for this dissertation, demonstrating that one can produce models with them.

- **Assumption:** The Comparison technique of DESC assumes the existence of a retrieved, alternative model of the design model given as input for the technique to work. It furthermore assumes that, if the two models share they same terminology, they are literally referring to the same concepts with these terms. The alternative model could come from a knowledgebase of designs, it could come from a design teammate, or (if only considering DESC in a controlled, experimental setting like in this dissertation) it could come from some oracle. Regarding the design teammate being a source for alternative models, this is particularly appropriate in the interdisciplinary context of biologically inspired design where certain members of the design team may be experts in one field (e.g., biology) and not in another field also relevant to the design challenge (e.g., an engineering discipline). Retrieval of the SBF-family of models has been shown possible in prior works (e.g., [160, 172]).

These three assumptions relate to knowledge conditions in the sense that they are about the knowledge constructed/retrieved for and used by DESC. First, are SBF* models themselves reasonable? As a kind of functional model, SBF* models align with the context of conceptual design in which this dissertation is situated, for conceptual design entails creating a functional decomposition and so function is meaningful in this context. Thus, models that emphasize function, such as SBF*, should be reasonable for this context.

Second, is it reasonable to expect SBF* models to be constructed? This question can be broken into two parts. Part one, is it reasonable to expect a practitioner and their teammates to have the competency to construct SBF* models? Constructing these models is admittedly a non-trivial skill, but their existence in this dissertation suggests it is nevertheless possible to gain that skill. Furthermore, the modeling interface developed for the Usefulness Study explored how one might scaffold model-building (e.g., using an interface that consists of forms and buttons rather than writing in a formal language), suggesting that user interface/interaction work could be done to support model construction. In this spirit, such competency (with or without being aided by scaffolds) is reasonable in the long term.

Part two, is it reasonable to expect practitioners and their teammates to make the time to build SBF* models? Without knowing the specifics of a design team's working conditions it is difficult to answer such a question in any concrete sense, and this dissertation is admittedly limited in not situating itself directly in a design context (see the second limitation below). That said, two points can be made. First, the Comparison technique can theoretically be applied to models of various degrees of completeness (e.g., only functions; only structure submodels; partially defined behaviors), so only the desired amount of modeling could be done to leverage it. Second, one assumes that time is made for activities deemed valuable. If practitioners find the incremental model revision process that is facilitated by DESC to be useful in their workflow, it follows that they would then make time for constructing and refining the models needed to work with it.

An additional knowledge condition relates to the time and effort it takes for the practitioner to process the feedback given to them by DESC. This topic, while important for making DESC useful, is not addressed by this dissertation and remains a topic for future work to consider.

In addition to assumptions, the following are a set of limitations that constrain this dissertation:

- **Limitation:** The DESC AI agent developed for this dissertation focuses on a single functional modeling representation, SBF*. This means that any generalization to other functional model representations or other kinds of representations requires rethinking/adaptation of the approaches in DESC. That said, the general idea of simulation-based evaluation (if not in this exact form) has been shown to be a plausible approach to evaluation in prior works, and the use of comparison for evaluation could be quickly adapted to other structured representations–even Compositional Mapping only requires the ability to logically partition a knowledge representation.

- **Limitation:** The two experiments (see below) are conducted outside of a design context. Thus, the findings of this dissertation are limited in that they cannot claim direct impact on a design process (biologically inspired design or otherwise). A clear area for future work to strengthen this research would be to explore the use of DESC within an explicit design context as part of future work.

- **Limitation:** The Usefulness Study (see below) was conducted with a very small sample size for pragmatic reasons. This limits the ability to draw strong conclusions from the findings. It would strengthen the work to conduct the study with a larger sample size to determine what findings represent legitimate, statistically significant patterns.

- **Limitation:** Although most of the models constructed by this dissertation were derived from real-world cases of entities thematically or directly related to biologically inspired design, the dissertation author was either solely or collaboratively involved in constructing all of them, which inserts bias and, for those models where the author (neither a biologist nor a domain expert) solely constructed the model, inserts error and lack of detail. Future work that conducts deep analysis on models constructed entirely by others (especially if those others are domain experts) would also strengthen this work.

- **Limitation:** Alongside the prior limitation, the set of model differences and problems described in Chapter 4 are the product of speculative thinking given an understanding of SBF* models. They may not be complete or pragmatic. Future work that looked at a representative sample of designer-constructed models to derive empirically-grounded taxonomies of model differences and problems would better inform areas to address with model evaluation.

- **Limitation:** The Usefulness Study's analysis does not occur on a fine-grained enough level to directly inspect any potential incremental model revising that occurred during external model building, so one can only speculate about the results' relationship to incremental model revision from a higher level.

- **Limitation:** Straddling the line between an assumption and a limitation is DESC's tacit assumption that behaviors in SBF* models consist of a linear sequence of states connected by transitions where a given state has at most one incoming and one outgoing transition. Although not experimentally tested, consideration of DESC's two techniques suggests that the Comparison technique would be capable of handling nonlinear behaviors. However, Simulation has technical and theoretical limitations that prevent it from doing so. A more nuanced view of this limitation with respect to Simulation is discussed in the Conclusions chapter (Chapter 7) along with ideas for how one might enhance SBF* (and implicitly also Simulation's reasoning) to handle nonlinear behaviors.

1.5 Scope: Knowledge, Task, and Domain

This section frames the scope of this dissertation: the knowledge it operates over, the task it addresses, and the domain in which it is contextualized. This dissertation operates over **knowledge in the form of well-formed SBF* models**, where well-formed here means that they are assumed to be complete in terms of having a structure, behaviors, and functions; they are syntactically correct (with one exception); and they do not contain the simulation-breaking error of a loop in the function decomposition. These models could in theory be on any topic that an SBF* model can reasonably represent. This work demonstrates a range of such topics. The six example models (including the synthetic model) created for this dissertation are of technological systems, biological systems, and one physical phenomenon.

Zooming out from SBF* models, while the implementation of the Comparison technique of DESC is certainly tied to SBF* models, Comparison at a theory level should be capable of handling structured representations in general, assuming that the strategy is adapted to reason about the mapping between two representations and the output is similarly modified.

Focusing on the two mapping algorithms that were developed for Comparison: Uniform Mapping, for example, is a wrapper around the Structure-Mapping Engine [44], which operates over what are essentially semantic networks (a generic structured representation). Compositional Mapping is itself a meta-framework that can use any mapping technique as subalgorithms (e.g., a set of Structure-Mapping Engine instances), making it adaptable to different knowledge representations and thus requiring only that constraints can usefully be propagated between partitions and (for it to differ from Uniform Mapping) for the representation to be logically capable of being partitioned. Interestingly, one could even imagine Compositional Mapping operating over a multi-modal representation, dividing (for example) a structured part, image part, and natural language part into separate partitions with their own subalgorithms. The challenges here would be how to pass constraints between these partitions and how to reason about and represent the results.

The Simulation technique, on the other hand, is closely tied to SBF* models because, even at the theory level, it assumes certain features in the representation such as state-transition diagrams and functions. That said, prior research that uses simulation to evaluate knowledge shows that the general notion of using simulation for evaluation is amenable to other knowledge representations besides specifically SBF* models.

Specifically, this dissertation addresses **the task of evaluation in the broader task context of conceptual design for the purposes of enabling incremental design revision**. This follows from the evaluation of functional models (plausibly constructed during the conceptual design stage) and the intent to evaluate the candidate design, prior designs, and source cases (plausible knowledge entities in this phase) with the implicit goal that this evaluation will lead to better models and, in so doing, hopefully better internal understanding on the part of the practitioner. That said, Compositional Mapping is potentially applicable to the abstract, analogical reasoning task of mapping two knowledge representations since it need not be attached to the Comparison technique (that is, to evaluation).

This dissertation targets **the domain of biologically inspired design**. The models tested (except for the synthetic model) are thematically plausible in this domain as are the knowledge entities (candidate design, prior designs, and source cases) targeted by DESC. However, even assuming that all of DESC is tied strictly to SBF* models, one could conceivably broaden the scope to design writ large because SBF* models can generally model technological systems. This dissertation also demonstrates that SBF* models can capture biological systems, so one could also argue that this dissertation is broadly applicable to biology. That said, further research is needed to qualify/verify these claims and to determine what areas of design and biology are amenable to SBF* modeling (since it is unlikely that all areas are) and would benefit from the evaluation support provided by DESC.

1.6 Evaluation

Two studies were conducted in this dissertation to evaluate the Design Evaluation through Simulation and Comparision (DESC) AI agent and, in turn, test the hypotheses of this dissertation. One of these studies was a computational, ablation-like study referred to herein as the Computational Experimentation. The purpose of this experimentation was to test DESC's ability to evaluate models for internal and external consistency issues. Recall that evaluating external models relates back to the broader picture of supporting incremental design revision, as these external models (in that broader context) act as proxies for the internal models held by practitioners. The Computational Experimentation evaluated Hypotheses 2.1, 2.3, 3, and 4. In so doing, the Computational Experimentation also evaluates part of Hypothesis 1 indirectly.

The other evaluation done for this dissertation was a controlled, pilot study with human participants referred to herein as the Usefulness Study. The purpose of this study was to test if an implementation of DESC could help participants develop better models and better understanding of the modeling topics compared to those without DESC. In so doing, this work explored Hypothesis 1 by looking at DESC in an external model-building context.

Each of these two evaluation studies will be discussed in turn.

1.6.1 Computational Experimentation

The Computational Experimentation worked as follows. A series of trials were conducted– one set for the Simulation technique and one for the Comparison technique. In each trial, a technique implementation ran against one (for Simulation) or two (for Comparison) SBF* models given as input. The results (one dependent variable) were converted to human readable form and collected for analysis. The run times– plus notes taken when working with interaction– were also captured (the second dependent variable). The trials were manipulated through several independent variables: (i) which model was used, (ii) what was the configuration of the model, and (ii) what was the configuration of the technique. For that Comparison trials, one of the two models was always in the Original (unchanged) configuration and both models had the same topic (i.e., were derived from the same model), so (ii) refers to the configuration of the second model, which could have also been the Original configuration.

Six different models were used in the Computational Experimentation. They are code-named Train, Owl Wing, Synthetic Car, Medical Patch, Electric Toothbrush, and Friction Drag. One model (Owl Wing) refers to a biological system, one (Friction Drag) refers to a physical phenomenon related to a biological system, three (Train, Medical Patch, and Electric Toothbrush) refer to technological systems, and one (Synthetic Car) is a synthetic model involving a technological system. All six of the Original configurations of these models along with short descriptions of what each represents can be found in Appendix D.

For Simulation trials, all model configurations besides the Original configuration reflect possible modeling mistakes. These were as follows: putting the wrong attribute value in a state condition, putting the wrong attribute value in a function provides condition, and writing a bad equation. For Comparison, all model configurations besides the original configuration reflect different ways to represent the design. These were as follows: use different terminology, combine two components and their attributes into a single component, and integrate a subfunction and subfunction's behavior into the superfunction and its behavior.

Technique configurations generally represent, conceptually if not literally, ablated or non-ablated versions of the techniques. Each ablated version of Simulation is missing one of its major features, where the non-ablated (Complete) version has all of these features enabled. Comparison configurations differ in which mapping technique is used. It has two configurations for Compositional Mapping and one configuration using Uniform Mapping. Although Uniform Mapping is not technically an ablated version of Compositional Mapping, it acts as such because it is missing the major theoretical ideas of problem decomposition and constraint propagation that underlie Compositional Mapping. The two versions of Compositional Mapping differ in that one uses some subalgorithms that potentially allow for user interaction, whereas the other configuration does not ever allow user interaction. Instead of being viewed as another ablated configuration, the non-interactive version of Compositional Mapping exists to reduce confusion about the running time of Compositional Mapping due to time spent in interaction.

1.6.1.1 Results

For Simulation technique trials, the output was inspected to determine if the technique found the complete set of valid model issues (if there were any) and if it found any invalid issues. A valid issue is a Simulation-detectable modeling mistake based on manual analysis of the configured model (i.e., the version of the model in the given trial) informed by foreknowledge that the Original configuration of the model (i.e., the version of the model with any intentional alternations) is error-free. An invalid issue is any identified issue not in the set of valid issues.

The Simulation results **support Hypotheses 2.1 and 2.2** in showing that this technique can evaluate a functional model of a design through simulation-based reasoning, using a knowledge representation (SBF*) with syntax and semantics that support automated simulation. Only the Complete technique configuration (which had all of the major features of the Simulation technique enabled) always found all valid issues (if any) and never found any invalid issues across the full set of model configurations. The other technique configurations (each of which were missing one of the features) did not achieve this level of success across all of their trials, and two of the three other configurations had at least one trial where they ran into an error that prevented Simulation from running to completion.

For each Comparison technique trial, the output was inspected to determine if the technique identified any invalid differences between the two models given as input to the trial. One model, call it Model A, was always in the Original configuration, and the other, call it Model B, was in some selected model configuration. This model configuration could have also been the Original configuration (representing an identity comparison). The two models were always configurations from the same modeling topic.

The differences detected by Comparison were considered valid if they were directly related (i.e., involve the model elements) that were changed due to the model configuration or would be changed if they are in the model that was always the Original configuration (model A). Any identified difference that involves *only* model elements *not* in this category were considered invalid. The idea here was to get a baseline, objective performance measure for a technique configuration's success. A fundamental level of operation is for the technique not to identify any invalid differences (as defined above) since doing so necessarily reflects a mistake.

The results of Comparison trials show, first, that all technique configurations do reasonably well. Across the full set of eight trials, no invalid differences were detected by any technique configuration on five of them. On the three remaining trials, only a small number of invalid differences were detected relative to the size of the models. These results **support Hypothesis 3** that Comparison can evaluate functional models of design through comparison-based reasoning. Second, consider Compositional Mapping (with interactive subalgorithms and without) versus its conceptually ablated counterpart Uniform Mapping. Compositional Mapping produced marginally superior results: it bested Uniform Mapping on two trials whereas Uniform Mapping only beat Compositional Mapping on one trial.

These results, along with the general finding that all technique configurations did reasonably well, **partially support Hypothesis 4** that Compositional Mapping can be used to map two functional models for the purposes of comparison and (implicitly) that it is superior to a mapping technique without its core ideas.

Given that Hypotheses 2.1, 2.2, and 3 were supported and Hypothesis 4 was partially supported, the Computational Experimentation **supports Hypothesis 1** in that it showed evidence that DESC can evaluate the internal and external consistency (via Simulation and Comparison, respectively) of design models. This is an important part of the hypothesized incremental design revising cycle to which the DESC AI agent contributes.

Finally, running times were also recorded for the trials of the Computational Experimentation, with time spent in interactive prompts (for the Comparison trials that involved Compositional Mapping with interactive moments) excluded from these totals. Although only an informal view of the performance of DESC, the running times were sufficiently brief on non-exceptional computer hardware (a personal laptop) to suggest that DESC is feasible for use as a design support tool.

1.6.2 Usefulness Study

The purpose of the Usefulness Study was to see if an implementation of the Design Evaluation through Simulation and Comparison (DESC) technique would usefully help human participants construct better models and improve their understandings of the topics being modeled. In so doing, this experiment explored the extent to which (Hypothesis 1) an AI agent can support systematic and repeatable incremental design revision through checking internal and external consistency.

The Usefulness Study worked as follows. Participants were assigned to one of two modeling topics, either the Technological condition (an electric toothbrush) or the Biological condition (friction drag, which is a physical phenomenon related in its source materials to a biological system). Participants were also either in the Experimental condition (they had access to an implementation of DESC during modeling) or the Control condition (they did not have this access). These two assignments reflect the independent variables of this study. Participants were asked to build part of an SBF* model of their modeling topic, and they were also asked (among other things) to articulate their understanding of the topic before and after model-building. These reflect the major dependent variables of this experiment. Both external models and conceptual understanding (via written articulation) were tested.

To support the implementation of the Comparison technique, an alternative model was needed to compare against the participants' models for each of the two topics modeled. Gold standard models of these topics were developed in collaboration with two domain experts (one per modeling topic). These gold standard models were also used in the analysis of this experiment to help determine how good a participant-constructed model was.

1.6.2.1 Results

Models produced by participants were analyzed to determine if Experimental condition participants (those who had access to the DESC implementation) would produce better models than Control condition participants (who did not). To answer this question, the definition of a *superior* model was operationalized using multiple perspectives, such that model analysis took multiple perspectives. It looked at things like the size of the model (larger is better) and the similarity of the participant's model to the relevant gold standard model (more similar is better). The analysis showed that Experimental condition-produced models tended to be superior to Control condition ones across the set of analysis operationalizations.

Survey data was also analyzed. Some of this data yielded useful insights to improve the modeling interface used by participants and to improve the implementation of DESC. The surveys also acted as a pre-test and post-test. Here, the results were mixed. When participants self-evaluated their understanding of the model topic on a scale, Experimental condition participants saw larger pre-test to post-test growth than Control condition participants. However, participants were also asked to articulate their understanding of the modeling topic, and here, Control condition participants saw larger growth.

Overall, these findings **partially supported Hypothesis 1** regarding DESC's ability to support incremental design revision. While participants with access to the DESC implementation seemed to produce superior models compared to those without access, this did not translate into improved understanding. A larger, refined study would be useful to determine the extent to which the small sample size of the Usefulness Study translates into a reliable pattern of behavior, and it would allow for improvements to DESC and the modeling interface that may positively improve DESC's impact. The analyzed results only capture the end of the modeling and understanding process (sometimes in comparison with the beginning), so any incremental changes along the way are unfortunately missed. Thus, one must conceptually view the experiment as a single instance of the incremental design revision cycle. A refined study would conduct analysis throughout the process to inspect things at a greater level of detail.

1.7 Conclusions

This section briefly recaps the extent to which the hypotheses and thesis statement were supported by this dissertation. It also summarizes key contributions made by this dissertation. Contributions are defined into two categories: theoretical and technical. Theoretical contributions are those that contribute ideas in the target domains, whereas technical ones contribute artifacts.

1.7.1 Reflection on Hypotheses and Thesis Statement

Below lists the extent to which the hypotheses to the research questions in this dissertation were supported by the two evaluation studies previously described. All hypotheses were at least partially supported. Note that the specifics of why each hypothesis is supported is described more in the previous two sections on the Computational Experimentation and Usefulness Study.

- **Research Question 1.** How might an AI agent support systematic and repeatable incremental design revision?

  Hypothesis 1. It may do so through checking the internal and external consistency of design models.

  - While not perfect, this hypothesis is **supported** through both the Computational Experimentation and the Usefulness Study. The Computational Experimentation (see responses to Research Questions 2 and 3) demonstrated that an AI agent is capable of checking the internal and external consistency of externalized design models. While not directly observing the hypothesized incremental design revision cycle and instead looking at the outcome of a modeling session, the Usefulness Study provides preliminary evidence that the presence of such an AI agent can support production of superior external design models. However, the Usefulness Study did not show that this led to better internal models, so more work is needed to improve how the DESC AI agent supports incremental design revision.

- **Research Question 2.** How might an AI Agent check a design model for internal consistency?

  Hypothesis 2.1. It may do so by simulating the model and using the results to check the process and functional claims made by the model. The Simulation technique of DESC illustrates this hypothesis.

  Hypothesis 2.2. The SBF* functional modeling language provides the syntax and semantics needed for automated simulation.

  - Both hypotheses were **supported** by the Computational Experimentation, which demonstrated that the Simulation technique of DESC was capable of successfully addressing a set of model ablations relating to internal consistency issues using simulation. For Hypothesis 2.2, it is noted that the Simulation technique's automated simulation approach operates over the SBF* knowledge representation.

- **Research Question 3.** How might an AI agent check a design model for external consistency?

  Hypothesis 3. It may do so by analogically comparing a given model against another model of the same design, where differences represent potential issues. The Comparison technique of DESC illustrates this hypothesis.

  - **Supported** by the Computational Experimentation, which demonstrated that the Comparison technique of DESC was reasonably capable of not identifying invalid differences when comparing two models. This is a baseline capability for external consistency checking by analogical comparison.

- **Research Question 4.** Analogical comparison implies mapping. How might an AI agent map two models?

  Hypothesis 4. It may do so by decomposing the mapping task, solving the subtasks, and using solutions from earlier subtasks to constrain and inform solutions to future subtasks. Compositional Mapping illustrates this hypothesis.

  - **Partially supported** by the Computational Experimentation, which demonstrated that Compositional Mapping was a decent mapping technique and was slightly superior for the purposes of Comparison when compared to a more conventional analogical mapping technique illustrated by Uniform Mapping.

The thesis statement for this dissertation is as follows: *"The conceptual phase of biologically inspired design entails incremental design revisions across multiple knowledge entities. An AI agent may support systematic and repeatable incremental design revision by checking the internal and external consistency of functional models of candidate designs as well as biological source cases and prior designs."* This thesis statement is supported by this dissertation in two ways. First, the Computational Experimentation illustrates that an AI agent can check the internal and external consistency of candidate designs as well as biological source cases and prior designs. Second, the Usefulness Study provides preliminary evidence that access to such an AI agent may lead to superior external models of topics thematically related to biological and technological designs although this did not appear to translate to superior internal models. Thus, while DESC was not completely successful at supporting incremental design revision, it did produce promising results.

1.7.2 Theoretical Contributions

The theoretical contributions of this dissertation get to how it conceptually breaks new, valuable ground in related fields.

Biologically inspired design (BID) may engage knowledge of biological source cases, prior designs, and candidate designs (and possibly others not covered here). Each of these may contain errors. In the related conceptual design, case-based design, and analogical design fields, DESC is the first work, based on those closely surveyed, that evaluates all three of these kinds of knowledge.

In the BID field itself, DESC contributes to a small body of research on computational evaluation, with only two other existing computational (or computationally-informed) tools in this area based on a recent survey. DESC well suited to this domain and novel relative to its peers by uniquely evaluating (in addition to the candidate design) the prior designs and biological source cases used in designing, given that the interdisciplinarity of this field means a designer may not have expertise in one (engineering) or the other (biology).

DESC is the first work in design critics among those closely surveyed to explore both the source case and prior design in addition to the candidate design and to do both simulation and comparison approaches together. The design critics surveyed typically used rules for their evaluation, although one also used comparison in a way similar to DESC.

Regarding using simulation to verify models, the surveyed prior works typically use qualitative reasoning (although other works may use quantitative reasoning or combine quantitative and qualitative reasoning). This makes sense for early stages of understanding, but quantitative knowledge might also be useful as understanding grows. It would be helpful to evaluate that as well because that quantitative knowledge could also have issues. DESC supports both qualitative and quantitative knowledge and reasoning, which facilitates both the early stages of knowledge development (qualitative) and allows for intermixing or adding/replacing more detailed information (quantitative).

Existing work surveyed in this area tends to generate results for the complete model (or for the entirety of what part was simulated). This could become difficult to digest as models grows. Although DESC's internal simulation reasoning may incorporate multiple parts of the model, it organizes and presents the results on a per-function and per-behavior basis, which should help practitioners focus when cognitively processing results and is helped if they address issues from the lowest level behaviors and upward.

This dissertation also presents Structure-Behavior-Function Star (SBF*), a version of the Structure-Behavior-Function modeling language that was co-developed by the dissertation author. SBF* adds syntax and semantics to SBF (and via the reasoning algorithms enacted by DESC) to support automated simulation. The experimentation and model-building done herein illustrates the languageâĂŹs usability and representational capabilities.

This dissertation also contributes a method to computationally map two concepts. Compositional Mapping is novel compared to the closely surveyed techniques by decomposing the mapping problem into a series of subproblems to be solved in a coordinated (through constraints) fashion and where subproblem solvers can be tailored to the associated representation partition. Additionally, in developing the Uniform Mapping approach, this dissertation tacitly explores the utility of the Structure-Mapping Engine prior work (via Case Mapper's implementation) at mapping two SBF* models.

Overall, DESC is unique amongst all surveyed works in its combination of *what* it evaluates and *how* it evaluates them. No closely surveyed prior work evaluates all of the biological source case, prior designs, and candidate design using both simulation and comparison.

1.7.3 Technical Contributions

The technical contributions of this dissertation reflect produced artifacts. The contributions can be mostly divided along two categories: DESC and SBF*.

This dissertation contributes an operational implementation of the AI agent DESC (Design Evaluation through Simulation and Comparison), which facilitated this current research work and could support future work as well. The Simulation and Comparison techniques (and supporting code) process Structure-Behavior-Function Star (SBF*) models and produce human-readable output. Compositional Mapping and Uniform Mapping approaches are interchangeable parts of this implementation. As part of the implementation, an automated interface was developed on DESC's side to interact with Case Mapper's server interface.

Regarding SBF*, the dissertation author co-developed a formal language definition for this modeling language. This facilitates reproduction of this work and the language's use in other work. Six illustrative models in this formal language were also developed or co-developed. These may be used as examples in future work. A language definition used to represent parts of SBF* models in Case Mapper was also developed. It may aid future work where the Structure-Mapping Engine is used to map SBF* models.

Finally, combining these two categories, an operational web application for constructing SBF* models and getting feedback on them from DESC was developed for the Usefulness Study. This could support future experimentation and it could be refined and enhanced to test DESC and/or SBF* in, e.g., design contexts.

1.8 Structure of this Dissertation

The remainder of this dissertation is structured as follows.

Chapter 2: This chapter contextualizes this dissertation in related fields through discussing prior works in those fields.

Chapter 3: This chapter addresses the question of what is the knowledge representation for design models given as input to the Design Evaluation through Simulation and Comparison (DESC) AI agent and used for alternative models. It describes the specific instance of the Structure-Behavior-Function (SBF) modeling language used in this dissertation, which is called Structure-Behavior-Function Star (SBF*). SBF* acts as the knowledge representation that DESC operates over, and the models used in this dissertation are thus constructed in this language.

Chapter 4: This chapter addresses the question of how an AI agent checks external design models for internal and external consistency. It describes the Design Evaluation through Simulation and Comparison (DESC) AI agent. Within this chapter, the two techniques of DESC, Simulation and Comparison, are described in detail.

Chapter 5: This chapter describes the Computational Experimentation that was conducted to test DESC's ability to evaluate the internal and external consistency of designs.

Chapter 6: This chapter describes the Usefulness Study, a controlled, pilot experiment conducted with human participants to test whether an implementation of DESC can improve a person's ability to construct models and understand the topic being modeled, exploring DESC's usefulness in the hypothesized incremental design revision cycle.

Chapter 7: This chapter reflects on the extent to which this dissertation's hypotheses and thesis statement are supported by the results of the Computational Experimentation and Usefulness Study. It also details the contributions of this dissertation and speculates about potential future work.

This dissertation also contains a set of appendices that represent supporting materials and explanations outside of the main research narrative. They are as follows.

Appendix A: This appendix presents the formal SBF* language definition. This language underpins the models in Appendix D.

Appendix B: This appendix presents the SBF* language definition for use with the Case Mapper external technology, which was used to apply an instance of the Structure-Mapping Engine during the mapping phase of DESC's Comparison technique. This language defines how to represent SBF* models when interfacing with Case Mapper.

Appendix C: This appendix presents recreations of the surveys and other materials used in the Usefulness Study along with descriptions of the source materials given to participants.

Appendix D: This appendix presents the SBF* models used in both the Computational Experimentation (in their Original configurations) and the Usefulness Study. Among that set, with their names slightly changed, are also the models used as the gold standard models and the model used during training in the Usefulness Study[5].

[5]Due to organization issues, it is possible that these models differ from those in the Usefulness Study, but they are believed to be the same.

CHAPTER II

RELATED WORKS

This chapter contextualizes the Design Evaluation through Simulation and Comparison (DESC) AI agent in prior, relevant research. In addition to presenting some high level sampling from related fields, this chapter also presents closer comparisons with a sample of relevant works. This chapter touches on how DESC differs from these works in terms of either taking a different perspective on how to do evaluation, on what to evaluate, or on both, thus illustrating DESC's novelty in various fields. These closer comparisons tend to focus on computational works explicitly about evaluation or that contain evaluation aspects. Fields covered are conceptual design, case-based and analogical design, biologically inspired design, analogical reasoning (in the abstract), functional modeling, and design critics. It also touches on SBF modeling, and it looks at prior computational work in analogical mapping, which is relevant to Compositional Mapping. At the end, a few other related fields are also covered.

2.1 Conceptual Design

Design as an activity involves developing a solution, typically an artifact, to a design problem. [46] surveyed methodology and design theory in mechanical design. [23], which proposed a "task structure" for this field, is an example of prior research in the domain. Other prior works in design: [40] brought together the fields of design and AI. [139] related AI to "design synthesis." [151] included a retrospective and discussion the future of AI in computer-aided design via "intelligent CAD." [143] is an example of computational work in design about resolving bugs in electrical design. [103]'s BOGART in the VEXED editor conducted partially automated circuit design and redesign by replaying a design decision history. [17] is about how to build "knowledge-based expert systems" to do "*routine* *design*". [141] set up a relationship between a human and machine in design, with the human making decisions about control that the machine enacts. [97] relates to this thesis in that they developed a sort of automated incremental design revision system in the context of elevator design. [140] developed a means to automatically generate new designs from a sketch of a mechanical device. [41, 81] conducted work related to design databases, [146] described a design repository, and [104] described a modeling language for use in design databases. [33] described a "concurrent engineering infrastructure" called PACT that contains technologies to support design. The paper discussed initial experimentation on "distributed simulation" and "incremental redesign." [47]'s work, which includes the development of a conceptual framework, is on supporting design team collaboration when there are communication challenges through developing design environments. [73] looked at analogy use for problem evolution, with a case study from biologically inspired design where they inspected "the evolution of an ill-defined design problem from inception to conceptual design."

More in the realm of social science research, [1] conducted a study to look at how novice and expert designers operate, and [19] also looked at this dichotomy but in terms of how it relates to "visual analogy." [102] looked at the difference in design knowledge representation between freshman engineering students and senior engineering students. [4] studied how a designer uses mental imagery. [145] looked at how sketches are conceptualized in architectural design. [144] developed a scheme for analyzing cognition in design, and they used it to a investigate an architect's cognitive interaction with their sketches. [13] did two studies. One looked at "the role of analogical in creativity" (specifically at a more narrowly defined aspect of it), and the other looked a topic related to designers of different expertise levels. [32] discussed how to teach/improve design-related skills. [79] investigated design fixation. [111] developed a model of re-representation in design. [165] looked at how and when an engineer deviated from their plan during a functional specification task and proposed an explanatory framework for this behavior. [87] conducted protocol studies of designers and identified four cognitive strategies that they use. [152] looked at the impact on idea generation by when one receives information and how similar that information is to the design problem. Although not specifically in the field of design, [99] looked at the performance of different kinds of "colocated new product development (NPD) teams." This work relates to collaboration, which is a relevant topic in design.

Conceptual design [112] is an early stage of the design process where one initially develops a solution to the design problem. As examples of work in this field: [92] developed a tool to support problem formulation in this stage. [169] developed a behavioral representation and associated computer program implementation to support conceptual design. [2] used machine learning to learn design rules for conceptual design. [68] developed software to map symbolic descriptions to descriptions in geometry. Although not about conceptual design, [135] also took a machine learning perspective on design, developing a formalism, as they call it, that answers learning questions such as how and when learning occurs.

The Design Evaluation through Simulation and Comparison (DESC) AI agent for design evaluation targets the conceptual phase of design. The following discussion samples the evaluation aspects of some computational works in conceptual design to contextualize DESC in terms of prior work. Table 2.1 presents a high-level summary of the surveyed works relative to DESC.

Klenk et al.'s [85] work combined qualitative simulation with Modelica models of designs, verifying functional requirements against simulation results drawn from topologies. While not specifically targeting conceptual design, Klenk et al. targeted an "early" stage of design that occurs before detailed design, hence their classification here. DESC also uses simulation to evaluate designs, but this is done as part of a two-part process that also uses model comparison to get a different perspective on evaluation so that issues that may have been missed using simulation alone can also be caught.

Komoto & Tomiyama [86] developed the SA-CAD system to support systems architecting for multidisciplinary systems. SA-CAD enables a system architect to develop an

Table 2.1: Comparison between DESC and conceptual design works

Research Work	**Evaluation Target**	**Evaluation Method(s)**
Klenk et al. [85]	Candidate design	Qualitative simulation
Komoto & Tomiyama [86]	Candidate design	Qualitative simulation; KB-based inferencing; Model-based comparison
D'Amelio et al. [35]	Candidate design	Difference detection supported by qualitative reasoning
Chin & Wong [29]	Candidate design	Heuristic scoring; User feedback
Umeda et al. [154]	Candidate design	Qualitative simulation
DESC	Candidate design; Source case; Prior design	Qualitative simulation; Quantitative simulation; Model-based comparison

SA-model consisting of a metamodel (a Function-Behavior-Structure model) and a parameter network (a detailed view of the behavior and structure). The system architect can also develop geometric and temporal models of the design. Evaluation occurs in two places in this work. First, the metamodel is an FBS model that can be evaluated through qualitative simulation or "Physical Feature-Based Reasoning", which seems to be inferencing using knowledgebase entries. Second and more the work's thrust, the system can automatically compare spatial and temporal relations in the metamodel against those in the geometric and process models, respectively; guarding against inconsistencies created by (perhaps) people in different disciplines. Komoto & Tomiyama's work is similar to DESC in the use of both simulation and comparison approaches for evaluation. That said, the specific methods for conducting these techniques differ between the two works (e.g., DESC does quantitative and qualitative simulation whereas SA-CAD does only qualitative; DESC does comparison between two models of the same type using analogical mapping whereas SA-CAD's comparison is between models of different types by analyzing values communicated to a third type of representation). Further, DESC evaluates the prior design, source case, and candidate design whereas SA-CAD only explicitly targets what this dissertation terms the candidate design, which is somewhat expected given that this work is not contextualized in case-based or analogical design. This is nevertheless a gap in research left open by this work. Komoto & Tomiyama's use of knowledgebase entries to support inferencing for evaluation is not comparable to anything in DESC.

D'Amelio et al. [35] provided a computational mechanism to verify Function-Behavior-State models (see [155]) in the context of conceptual design (or, at least, an early stage of design). Specifically, they used qualitative reasoning to infer the extent to which a design (either a redesign or an integration of modules) produces unpredicted behaviors or is missing predicted behaviors. This brings awareness to the designer and allows them to determine if these behaviors are undesirable and thus should be fixed. Although D'Amelio et al.'s contrast filtering method (that compares between an old design and a new design) does model comparison like DESC's Comparison technique, there is a subtle difference in meaning. DESC compares two models that are supposed to be of the same topic, so differences detected by DESC suggest that one of the models is in error and should be corrected to align with the other model. D'Amelio et al.'s comparison is between two known-different topics (an old design and a new design), so unpredicted differences here represent consequences of that known difference that may need to be addressed without implying that one should necessarily revert the new design to the old design, for presumably the new design exists for some purpose. Given this difference between the comparison approaches and given D'Amelio et al. not having anything like DESC's Simulation technique, one concludes that the kinds of errors searched for by [35] and those by DESC are different: D'Amelio et al. looked for issues due to redesign or integration, whereas DESC looks for issues that they would not catch: errors in understanding and representation.

Chin & Wong [29] developed the EIMPPLAN system to automate the conceptual design stage in the domain of plastic injection molding. Their goal was to address all stages of the conceptual design process in this domain. Specifically, in this work, they discussed the implemented EIMPPLAN-1 system that addressed (with user input/feedback) material selection and mold design feature generation. It is unclear exactly how the authors circumscribe the evaluation aspects of their work, so for the sake of discussion, two methods are identified here: (1) heuristic scoring such as defining features of materials as "SU (superior), SA (satisfactory)" etc. and using these scores for selection and ranking, and (2) direct human user feedback. Both of these approaches differ from how DESC evaluates designs, and so DESC thus takes a different perspective on evaluation.

Finally, Umeda et al. [154] developed the Function-Behavior-State Modeler or FBS Modeler to support conceptual design by enabling users to develop FBS diagrams. Once the model has been built, the FBS Modeler can evaluate the model through qualitative simulation of the model's behavior (which also can be considered as using the functions since functions add physical phenomena to the behavior), identifying physical phenomena that do not occur in the simulation, any unexpected phenomena that did occur, and any modeled functions that cannot be realized. In this way, Umeda et al. evaluated both the behavior and functional aspects of their models through simulation, similar in intent (if not execution) to the Simulation technique of DESC. That said, DESC's Comparison technique takes a novel perspective to evaluation relative to this work.

2.2 Case-Based Design / Analogical Design

Case-Based Design is a design process whereby one retrieves and (if necessary) adapts/repairs a prior solution to solve (or support solving) a current design problem. [171] proposed behaviors in design for case-based reasoning to support. It also proposes extensions on the case-based reasoning framework and an in-development architecture they are using for experimentation. [94] presented a survey of Case-Based Design computational works (including commentary on how they perform evaluation). [62] also surveyed this field, including challenges posed by design to case-based reasoning (the broader term for this reasoning paradigm). [93] is an in-depth look into the topic. [95] surveyed works of case-based reasoning applied to design. [56] also related case-based reasoning to design. [180] developed a system that, although they speak in terms of analogy, resembles case-based reasoning to aid structural design. [51] also developed a case-based system to support design. It included the ability to automatically store solutions as cases and the ability to retrieve them. [77] used constraints to support case adaptation in case-based design. [91] interestingly used a genetic algorithm seeded by past cases to generate designs. [37] addressed case adaptation using ideas from genetic algorithms; their case-based reasoning work is relevant here because they apply it to the domain of layout design. [138] discussed using multiple cases in hierarchical case-based reasoning in the context of plant-control software design. [16, 136, 137] are additional example works on case-based reasoning in or applied to design.

Analogical design is a related process, defined here as using knowledge extracted from one or more source cases (perhaps in abstracted form) to develop the design solution rather than using the source case(s) directly. [10] explored automated retrieval in this context. [90] conducted an experiment to better understand aspects of design and analogy. [36] developed a computer program that models human experimental participants doing analogical design. [58] presented a small survey of the field, including a AI perspective. [159] discussed the interaction between problem decomposition and analogical transfer in design that leads to the generation of compound solutions. [22] looked at the relationship between modality, commonness, and distance on ideation when using analogies. [34] investigated the influence of analogical thinking on idea generation. [31], although not specifically about analogical design *per se*, studied the functions of analogy in design.

Being contextualized in biologically inspired design (itself a kind of analogical design), DESC is clearly related to the field of analogical design. DESC is also related to Case-Based Design both because case-based reasoning is a similar field to analogical reasoning (and the relationship should hold when adapted to design) and because DESC can evaluate any prior solutions that a designer may incorporate into their design process, which may have arrived due to a case retrieval-like process. Cased-Based Design is also an example field where computational tools have been developed to support practitioners. A sample of computational works with evaluation aspects in these fields is surveyed here to contextualize DESC in terms of prior work. Table 2.2 presents a high-level summary of the surveyed works relative to DESC.

Goel & Bhatta [66] presented a theory of analogical design called model-based analogy or MBA. In MBA, design patterns in the form of generic teleological mechanisms or GTMs are extracted from behavior-function models (a subset of their version of Structure-Behavior-Function (SBF) models) through a difference-comparison process. These GTMs become abstract source cases that are retrieved and transferred during reasoning to transform retrieved, insufficient, prior designs into candidate designs. Goel & Bhatta's work applied qualitative simulation to evaluate the candidate designs in a process that sounds similar (tracing the behavior) to the Simulation technique of DESC but is not well elaborated in this article. One could also view how MBA checks if the function of the retrieved prior design matches the desired function as a form of evaluation, but this is a stretch because MBA is not evaluating if the retrieved prior design actually does achieve its function. DESC differs from MBA in its approach to evaluation by additionally conducting comparison-based evaluation and by targeting the prior design and source case in addition to the candidate design, two knowledge entities that could also contain issues due to misunderstanding or lack of understanding.

Goel et al. [67] developed a case-based design system called Kritik2. ([63] also discusses this system specifically in the context of model-based diagnosing for adaptive design and redesign.) Kritik2 leveraged cases in the form of SBF models for its reasoning process, whose goal is to find a solution to some design challenge represented as a desired function. Evaluation comes into play when the system attempts to repair a retrieved design. Kritik2 will attempt to apply a repair strategy and then will evaluate the model by simulating (how is unclear) the effected behavior and determining if the behavior achieves the desired function. DESC differs from Kritik2 by also doing comparison-based evaluation to take a different perspective on evaluation and by evaluating the source case and prior design in addition to the candidate design.

Pearce et al. [114] described their system called Archie that supports designers in architectural design by enabling them to retrieve office building design cases, propose designs, and critique designs. The office building design cases in Archie are represented by a large number of possible features. Archie facilitated design critiquing (interpreted here as a synonymous activity to evaluation) by letting users select domain models (and presumably view them, although the paper also says these models were not viewable) and allowing users to comment on cases. Archie also facilitated design critiquing by automatically classifying a candidate design and showing similarly-classified cases to users, which one supposes in turn enables them to judge the proposed design by analogy to the retrieved cases. DESC differs from Archie by evaluating the prior design and source case in addition to (like Archie) what this dissertation terms the candidate design. DESC and Archie have very different evaluation strategies. Archie facilitated user feedback on designs by providing a means for comments, by (maybe) showing the users domain models, and by automatically retrieving similar cases that the user can use for critiquing by analogy (i.e., the goodness of a retrieved case transfers to the candidate design). Conversely, DESC uses simulation and model comparison approaches to evaluation, with model comparison in DESC being an apples-to-apples comparison between two models of the same topic to discover differences. DESC also evaluates the source case and prior design, which are not evaluated in this work.

Barber et al. [8] developed the AskJef system to support software engineers in designing interfaces. AskJef facilitated case-based design by allowing users to search for (via a problem specification) examples of past interface designs. Cases are evaluated through whether they are positive or negative examples of guidelines or (although it is not stated as such in the article) prototypical errors. This is a form of heuristic or rule-based evaluation. These examples play the role of prior designs[1] in the taxonomy established in this dissertation. Thus, DESC differs from AskJef both in its approach to evaluation (simulation and comparison instead of heuristics) and in what it evaluates (additionally evaluating the candidate design).

Table 2.2: Comparison between DESC and analogical / case-based design works

Research Work	**Evaluation Target**	**Evaluation Method(s)**
Goel & Bhatta [66]	Candidate design	Qualitative simulation
Goel et al. [67]	Candidate design	Simulation
Barber et al. [8]	Prior design	Heuristics
Pearce et al. [114]	Candidate design	Facilitates user feedback
DESC	Candidate design; Source case; Prior design	Qualitative simulation; Quantitative simulation; Model-based comparison

2.3 Biologically Inspired Design

Biologically inspired design (BID), biomimicry [9], or bionics [89] is (essentially) a design process whereby solutions are generated by taking inspiration from natural systems. In other words, BID is analogical design where source cases come from nature. For example, McKeag [100] described how a train was redesigned to reduce the noise it made by inspiration from both the skull of a kingfisher and the way owls fly silently (this is the source for the example in Chapter 1). DESC is contextualized in biologically inspired design to ground it in a real-world design process.

Prior research (see [7, 61] for collections of works) has analyzed/studied/reviewed BID and related topics [3, 15, 72, 88, 96, 133, 156, 158, 167], explored it pedagogically

[1]Whether to refer to the example cases in AskJef as prior designs or source cases should not be taken here as a strong commitment. The point being made is that AskJef evaluates something other than the candidate design.

[18, 178, 177], developed techniques and approaches to support BID practitioners [20, 21, 28, 30, 39, 107, 105, 106, 124], or worked to make existing techniques relevant to the domain [163] or used to study this domain [164]. [132] combined functional modeling with this field. [49] is yet another kind of work related to this discipline. Fish looked at drag reduction mechanisms in nature in comparison to those in engineering. In [50], Fish et al. highlight humpback whale flipper tubercles as bio-inspiration for wing-like designs. [11] explored biological phenomena and their applications. [109] is an example of biologically inspired design: the authors developed a server allocation algorithm by looking at honey bee forager allocation. As another example, [147] discussed a “biologically inspired mathematical model” of “adaptive network formation.” [98] discussed a “hydrogel-capped hair sensory system” that was inspired by biology. [55]’s work included, among other things, a look at the current state of the field and speculated on how it might address a set of societal challenges. [127] studied transfer in biologically inspired design. Their work included a representation that they felt could be useful for “modeling complex biological systems and their functionality.”

Based on a review of 43 tools made for biologically inspired design [168], BID only has two prior works on evaluation: [71, 170]. (Although, in a contradiction, [168] also seems to suggest there is only one.) T-charts [71] enabled analogy evaluation (meaning the aptness of a biological source) by having users align a biological source against the design problem across four conceptual categories called a Four-Box specification. T-Charts are not automated, but Helms’ work comes from a computational perspective. After filling out the T-chart, the designer determines the extent to which the contents of the two sides of the T-chart are similar to each other.

A high-level comparison between Helms’ work and DESC appears in Table 2.3. DESC differs in two important ways from this work. First, DESC is a (mostly) automated technique, whereas a T-chart, while useful, is a manual tool. This matters because DESC (except for the human interaction component) can facilitate repeatable, consistent evaluation through automation. Second, DESC evaluates the biological source case, prior design (if it exists), and candidate design. T-charts evaluate the relationship (or mapping) between the design problem and the biological source case. Given this last point, DESC and T-charts could be complementary approaches because they do not overlap.

The biotransferability framework developed in [170] is a computational work on evaluation for biologically inspired design that calculates confidence in whether a potential biological analog is a high, medium, and low transferability risk. Humans create scores for a given source against a set of heuristics such as how mature is knowledge of the source's field (factoring in how much does the rater know) and if the solution is based on extraction of principles or simply transferring what the source already does. The technology developed by the authors then automatically calculates confidence scores for each of the three categories. People can then use these results to decide whether to prune away potential sources. DESC takes a very different approach to evaluation than does [170]. Rather than using heuristic measures to calculate confidence values, DESC evaluates a design using simulation and model comparison. Williams et al. is fundamentally about evaluating the mapping of the biological analog, so its focus differs from DESC that targets the biological source cases, prior designs, and candidate design. That said, this work does look at evaluation from a different perspective and thus would likely catch different issues than DESC. Thus, like with [71], DESC is complementary to [170] and the two could be used together to broaden one's evaluation perspective.

Zooming out, the rarity of computational work explicitly on evaluation in this domain suggests that it is ripe for research in this area to both understand how to do evaluation in biologically inspired design and to develop tools for doing so.

2.4 Analogical Reasoning

Analogical reasoning is the process by which one infers knowledge about a concept (the target) by comparison to another, typically better known, concept (the source). Analogical

Table 2.3: Comparison between DESC and biologically inspired design works related to computational (or computationally-informed) evaluation

Research Work	**Evaluation Target**	**Evaluation Method(s)**
Helms [71]	Mapping between design problem and source case	Manual comparison
Williams [170]	Mapping between design problem and source case	Computed score from manual heuristics
DESC	Candidate design; Source case; Prior design	Qualitative simulation; Quantitative simulation; Model-based comparison

reasoning relates to this dissertation through being a more abstract form of the process underlying analogical design. [75] discussed a theory of analogy in human thought applicable to multiple domains. [121] studied the use of analogies in an education setting. [166] described two different uses of analogy in design. [5] also studied use of analogies (in addition to "mental simulation") in design. [148] conducted analogical retrieval through constraint satisfaction. [175] used constraint satisfaction for retrieval and mapping in analogies involving visual knowledge. [53] presented a two-stage process (non-structural then structural) for retrieval in analogical reasoning. [142] discussed multiple aspects of analogical reasoning, including analogical reasoning's relationship to "general intelligence." [69] looked at children's ability to solve analogies earlier than previously supposed by presenting problems with causal relations that they could understand. As an example of analogical reasoning outside of design, one can consider [128]'s work. This work was on generational experience, but it raises an interesting point that one's experience of this kind effected which historical analog among two was selected as most relevant to compare with the Iraq war, which in turn related to whether one supported (or not) that war.

Forbus et al.'s [52] Support metric evaluated analogical inferences. Support asks, "How much structural support does an analogical inference derive from the mapping that generated it?" Support is a calculated value that is used to judge the analogy. Forbus et al. presented another metric called Extrapolation, which also evaluated the analogy's inferences. Extrapolation asks, “How far does an analogical inference go beyond the support lent by the mapping?” Like Support, Extrapolation is a calculated value that is used to evaluate the analogy.

In the same work, Forbus et al. also discussed representing analogical inferences as propositions so that they can be used by other reasoning mechanisms. In the cited work, they appear to use this capability in an evaluative role when discussing an example where they analyze an inference through a “qualitative physics system” and discover a contradiction. The authors also discussed an example that involves pruning a set of analogies based on if the inference(s) generated by that analogy are relevant to the goal of the analogical reasoning process. If this approach is intended to transfer to other situations, this is another case of evaluation in analogical reasoning.

Falkenhainer’s [43] work evaluated the result of analogical reasoning. Falkenhainer describes evaluation through mechanisms that he terms “Consistency Verification” and “Empirical Verification.” In this work, one role played by analogical reasoning is to generate a new model about a domain, with heat flow given as the example. Consistency Verification entails comparing the new model against observations. Empirical Verification is only speculated, and it is to be ran on models that are consistent. With Empirical Verification, Falkenhainer proposed to run experiments to evaluate the new model.

When viewed as a computational theory for evaluation in analogical reasoning, DESC is similar to Falkenhainer’s [43] work in that it uses simulation to evaluate the concept generated by analogical reasoning. However, DESC’s Simulation technique supports both quantitative equations and functional reasoning in addition to qualitative reasoning, and the Simulation technique does not need ground truth to provide evaluation feedback. Looking at the Comparison technique, one could view the alternative model (that which gets compared against) as a proxy for ground truth (although it need not be), but here the comparison is done between the models, not involving the simulation output as it is in Falkenhainer’s work.

DESC does not have any overlap with Forbus et al.'s [52] work since it does not look at individual inferences and chooses a different perspective on evaluation (identifying issues) compared to their approach using numerical measures. The two works thus would be good complements given their lack of overlap.

Similarly (but from the other side of the coin), DESC uniquely evaluates the reasoner's understanding of the source and target concepts, whereas the prior works tend to focus on the outcome of the reasoning process: the entire produced concept or the individual inferences. Although the outcome is indeed important and is also covered by DESC, evaluation of the source and target concepts is vital for knowledge-based analogical reasoning since the process uses these concepts and may thus produce errors or otherwise negatively influence the outcome if they are erroneous or incomplete. That said, DESC does not evaluate the mapping between the source and target concepts or look at the analogy overall, topics suggested by [44] (see below).

Table 2.4 summarizes some of the above discussion and compares DESC, at a high level, to the two sampled prior works.

Table 2.4: Comparison between DESC and analogical reasoning works

Research Work	**Evaluation Target**	**Evaluation Method(s)**
Forbus et al. [52]	Analogical inferences	Heuristic measures; compared to goals
Falkenhainer [43]	Result concept	Qualitative simulation
DESC	Result concept (Candidate design); Source case; Target concept (Prior design)	Qualitative simulation; Quantitative simulation; Model-based comparison

Finally, the following article is given as an honorable mention. Although their article is mostly about the mapping phase of analogical reasoning, Falkenhainer et al. [44] presented three kinds of evaluation criteria. The below quote refers to a stage of analogical reasoning.

"(3) *Evaluation and use*: Estimate the 'quality' of the match. Three kinds of criteria are involved... The *structural* criteria include the number of similarities

> and differences, the degree of structural similarity involved, and the amount and type of new knowledge the analogy provides via the candidate inferences. The second criterion concerns the *validity* of the match and the inferences it sanctions. The inferences must be checked against current world knowledge to ensure that the analogy at least makes sense, and may require additional inferential work to refine the results. The third criterion is *relevance*, i.e., whether or not the analogy is useful to the reasoner's current purposes." [44, p. 4, emphasis theirs]

These guidelines appear to focus both on analogical inferences and the goal of the analogy. Evaluating the analogy with respect to the reasoner's goals (also mentioned in [52]) would make for useful complement to DESC especially given the design context. However, one could view DESC evaluating the extent to which the behaviors of the candidate design produce output consistent with its functions as evaluating with respect to goals since fulfillment of the design's functions is a meaningful part of design.

2.5 Functional Modeling

The prior sections in this survey focused on processes. Here, the discussion centers around content. Functional models or functional representations are a type of knowledge representation that emphasize function as an important aspect of the topic being modeled. Note that [26] viewed "functional representation" (which is perhaps a more technical term than what is meant by functional *representations*, plural) as a separate yet complementary thread of research from "functional modeling." As examples of work in this area: [83] developed a framework to capture and share functional knowledge, including in the form of functional models among other knowledge types, in conceptual design. [115] developed a functional representation and corresponding computer program that includes the device's environment in the model. [120] looked at hierarchies of functional abstractions and how they relate to design and elsewhere. [129] explored reasoning with a functional representation. [27, 25] discussed the FR functional representation, with the first citation including discussion of its limitations and the second including discussion of its applications, including the potential for “forward simulation.” [24] is also about FR; it is a ten-year retrospective that includes discussion of the representation’s applications. [153] discussed the FBS (Function-Behaviour-State) diagram, another modeling framework that includes function. [57] is another work in the field and has a different functional representation. [179] developed a functional modeling approach to determine what behavior modules are shared and unique among a product family. [108] added a grammar to represent signals in the Functional Basis representation. Although slightly off-topic, [84] did an interesting work where they automatically identify function information from behavior and connection information.

One way prior researchers have pursued evaluation of functional models (where here functional models is construed as models where function is a meaningful part of the representation, regardless of if the authors themselves term their work as such) is through qualitative simulation. Price [116] used “functional labels” to analyze the results of a qualitative simulation derived from a component model in terms of these functions as a means to simplify and organize the output to human reasoners. While Price’s work does not provide an account for design verification *per se*, it does analyze a model to analyze failures and sneak circuits (where something gets triggered unexpectedly) which is at least thematically similar to evaluation. Iwasaki et al. [78] verified functions by simulating a component model with additional behavioral pieces and comparing resultant trajectories against a function description that includes behavioral descriptions. (As an aside, [80] facilitated evaluation through automated simulations without using a functional model, illustrating that other knowledge representations can also be used with this approach.)

DESC’s Simulation technique differs from these methods in two significant ways. First, it preserves the hierarchical nature of SBF* models by independently performing a simulation and evaluation pass for each function-behavior pairing in a model. Other models may have abstraction built in (Price's [116] "[e]ncapsulate complex behavior within a component" or their use of function labels to simplify the output; Iwasaki et al.'s [78] model fragments), but their simulations and thus verifications produce results for the entire model in one chunk or otherwise, like with [78], are producing output for a specific aspect of the model. Simulation enables individual verifications to stay focused on one function or one behavior at a time even if the functional decomposition is very large while not requiring that a user scope the verification themselves. This automatic focusing should make results to handle compared to those that produce output for the entire model. Additionally, DESC is a step towards the "[m]ulti-level modelling" mentioned by Price et al. [117] as necessary to achieve future targets for qualitative reasoning.

Second, DESC leverages the same causal process representation (state diagrams, a.k.a., behaviors) in SBF* for both reasoning and representing simulation results, whereas other work uses representations for reasoning (e.g., component models or model fragments) that differ from the state-based representations in their simulation results. From the perspective of SBF* modeling, DESC thus does not require modelers to learn a new representation for reasoning since it leverages aspects of the models that are already part of the SBF* models being built–with the exceptions that modelers will need to learn equation syntax and simulation semantics (although the latter should be generally intuitive relative to modeling expectations). Additionally, that the verification results are couched in terms of the already-modeled elements (i.e., as issues in state conditions and function provides conditions) should lessen the cognitive burden relative to other systems because users do not need to learn a second representation to interpret the results as they would if one built a component model and then interpreted state-based representation results.

DESC also differs from [78, 116] by incorporating functions into the simulation. They impact the simulation process by acting as pointers to subbehaviors rather than just being used in evaluation/output. This empowers the simulator to use existing structural aspects of the model (i.e., the functional decomposition) in its reasoning, which should also, in the context of SBF* models, be similar to the way human modelers think about what the model is saying.

Finally, note that neither of these works conduct evaluation similar to the Comparison technique of DESC. DESC thus fills a gap in checking the accuracy and completeness (for its purpose) of the model relative to what is being modeled. Simulation-based methods appear to typically, tacitly assume that the models are already correct since they themselves do not know about the real world (i.e,. they need some connection to the ground truth, such as via a human reasoner, to break this assumption). Given that humans may hold incorrect/incomplete understandings of concepts in the world, this is an assumption worth checking.

Table 2.5 summarizes the evaluation target and methods for [78, 116] and places them in comparison with the answers for DESC.

Table 2.5: Comparison between DESC and functional modeling works

Research Work	**Evaluation Target**	**Evaluation Method(s)**
Price [116]	Candidate design	Qualitative simulation using function labels
Iwasaki et al. [78]	Candidate design	Qualitative simulation compared to detailed function structures
DESC	Candidate design; Source case; Target concept	Qualitative simulation; Quantitative simulation; Model-based comparison

2.6 Design Critics

Another field related to this dissertation is that of design critics. As an example, although it is not explicitly called out as a design critic *per se*, [122] developed a design environment for software architecture that includes critiquing as part of it. [134] discussed lessons learned from what they deem an incorrect approach to critiquing that involves "batch, after-task, debiasing of experts." They also advise about a better approach along with identifying "research needs" to improve future design critics in design support environments.

Fischer et al. [48], Bonnardel & Sumner [14], and Oh et al. [110] represent three example works from this domain that will be considered here in detail.

Fischer et al.'s [48] work on the HYDRA system in architectural (kitchen) design was a production-based system embedded in design software. Which production rules applied (i.e., which ones could fire if applicable) in a given design situation was decided by the perspective taken by the designer, the problem specification, or if it was a general design rule, with the former deciding factors taking priority over the latter.

Bonnardel & Sumner's [14] work was in the domain of automated voice dialog systems. This work studied how designers of different domain experience levels responded to feedback and interacted with the design critics, finding for example, that active critics were beneficial to all experience levels.

Oh et al.'s [110] work was in the domain of flat-pack furniture design. Their critic system also uses rules (although they use constraint terminology) to provide feedback to designers within a design environment. Oh et al. developed a system that tailors the kind of feedback (e.g., evaluative versus providing an example) and modality of feedback (e.g., text versus visual) to the model it has of the student designer using the system.

DESC differs from these three prior works in design critics in two ways: the analysis method and the target of analysis. Regarding the analysis method, the design critics in the prior work were largely driven by rules. DESC does not use rules; instead, its critiques designs through simulation and comparison approaches. Simulation is thus novel in this domain relative to these three sample works, thus exploring a novel means by which to generate critiques. Regarding Comparison, one notable exception to the use of rules in the prior works is the comparison approach in [14]. Here, the user's design is compared against similar designs to check consistency. This comparison approach is not described in enough detail for more detailed analysis, but conceptually, DESC's approach is similar. Despite conceptually similar approaches, DESC's use of comparison for critiquing (see below) differs in targeting design concepts other than the candidate design.

DESC also differs in the target of its analysis. Like the prior research, DESC targets the candidate design for critiquing. However, DESC also uniquely targets the designer's understanding of the source case (biological source of inspiration) and any used prior designs. In doing so, DESC expands the scope of design critics, relative to the three example works, to also consider other aspects of the design process beyond the design under construction. This should help prevent any errors in reasoning that result from misunderstanding of misrepresenting these two knowledge sources. At a theoretical level, it raises the question of what aspects of design an automated design critic should address, arguing that there is value to look beyond the candidate design.

Table 2.6 summarizes the above three related works and compares DESC to them, all at a high level.

Table 2.6: Comparison between DESC and design critic works

Research Work	**Evaluation Target**	**Evaluation Method(s)**
Oh et al. [110]	Candidate design	Rules
Bonnardel & Sumner [14]	Candidate design	Rules; Comparison
Fischer et al. [48]	Candidate design	Rules
DESC	Candidate design; Source case; Target concept	Qualitative simulation; Quantitative simulation; Model-based comparison

2.7 Structure-Behavior-Function Modeling

DESC specifically operates over Structure-Behavior-Function Star (SBF*) functional models. SBF* models were co-developed by the dissertation author as an iteration of the Structure-Behavior-Function (SBF) family of models [65, 59]. SBF models have shown their capability in representing biological and technological systems [162] and usefulness in reasoning tasks in the BID domain (e.g., [160, 172]), in analogical design [66], and to support case adaptation for innovative design [45]. They were also involved [157] in a system to help middle school students develop understanding of complex systems. Another prior work with SBF models is [60], where they were used to explain designs of devices. This occurred in the context of work that involved a computer system that addresses explanation by visually representing its reasoning and the design episode's result.

Through SBF*, this dissertation extends SBF modeling (mostly using [65] as a benchmark) by establishing syntax and semantics related to automated simulation. These extensions are presented in detail in Chapter 3 to keep them in context with the language definition.

2.8 Analogical Mapping

The next field discussed in this chapter concerns computational work on analogical mapping. Recall that analogical reasoning involves comparing a target concept against a source concept to make inferences about the target concept. To do this comparison, one must align, or map, the elements of the two concepts to know what elements in the source concept match what elements in the target and vice versa. Mapping is that task: matching elements between the source and target concepts. This alignment drives further reasoning (e.g., unmapped elements in the source can be transferred to the target as inferences). Analogical mapping relates to this dissertation because it explores a novel computational technique for doing so via Compositional Mapping. As an example of work in this field, [82] developed a unified framework for three computational models of analogical mapping and ran experiments to compare psychological performance against that of the models. [42] presented a "distributed model of analogical mapping" that worked on distributed representations. [123] looked at the relationship between similarity, mapping, and problem solving by analogy.

Here, three example mapping strategies are compared against Compositional Mapping. Table 2.7 summarizes the comparison between these strategies and Compositional Mapping.

One technique for analogical mapping is called the Structure-Mapping Engine (SME) [44], which essentially applies heuristic measures over a semantic network representation. Structure-Mapping prefers matches that involve higher-order (relationships between relationships, etc.) elements, it matches on relationships rather than superficial details, and it enforces a one-to-one mapping. Another technique for analogical mapping is called ACME [76], which is a connectionist approach. ACME represents the two concepts in a spreading activation network and decides which elements to map based on activation level after the network has settled.

Compositional Mapping differs from these prior works in two fundamental ways. First, both the Structure-Mapping Engine and ACME attempt to map both the source and target concepts all at once, whereas Compositional Mapping decomposes the mapping task into a series of sub-tasks, each mapping a logical partition of both concepts (according to boundaries in their knowledge representations). This simplifies the mapping problem by making each a smaller sup-problem (except in a rare edge case), which may lead to better solutions and more efficient reasoning. Second, prior work uses a single mapping algorithm, which makes sense given they map the models all at once. However, Compositional Mapping allows a different mapping algorithm per logical partition type. This in turn allows the implementor of a system using Compositional Mapping to optimize mapping algorithms at the partition level, which should lead to better solutions and may lead to more efficient reasoning if (for example) the complexity of the reasoning strategy can be made simpler for less complex partitions without compromising solution quality. Interestingly, because these subalgorithms are not themselves highly constrained by Compositional Mapping, the aforementioned SME and ACME approaches could actually themselves be used as subalgorithms.

[176] describe compositional analogy, which contains a multi-level mapping and transfer method in the domain of 2D line-drawings of physical artifacts (designs). Given a target that consists of a drawing and given a source that consists of a drawing and a structure (symbolic)

model, their work first maps at the level of lines and intersections in the drawings to transfer information about shapes. It then maps at the level of basic and complex shapes (learned from the previous step) to transfer symbolic information about the structure (components and relations).

Initially, Yaner & Goel's compositional analogy seems similar to Compositional Mapping (whose similar name is only coincidental) in that both address mapping in a decomposed manner–that is, in steps. However, deeper inspection reveals that the two methods are actually quite different in their approach. Compositional analogy is more akin to a series of (two) analogical reasoning episodes, with the transfer results of the first episode (at the line and intersection level) driving the second episode (at the shape level). Compositional Mapping, on the other hand, operates within the same reasoning episode, using mapping, but not transfer, results from previous steps to support future steps. Any transfer would occur after Compositional Mapping completes–although note that transfer in not explored in this dissertation.

In Table 2.7 compositional analogy is thus referred to as "Decomposed (Iterative)" in the third column because (as interpreted) it consists of two iterations of analogical reasoning, whereas Compositional Mapping is referred to as "Decomposed (Non-iterative)" because it all occurs before analogical transfer.

Table 2.7: Comparison between Compositional Mapping and analogical mapping works

Research Work	**Approach**	**All-at-once or Decomposed**
Falkenhainer et al. [44]	Symbolic	All-at-once
Holyoak & Thagard [76]	Connectionist	All-at-once
Yaner & Goel [176]	Imagistic & Symbolic	Decomposed (Iterative)
Compositional Mapping	Symbolic as implemented; flexible in theory	Decomposed (Non-iterative)

2.9 Other Related Fields

This section briefly looks at a few other fields that are related to this dissertation.

Unit testing is a method for evaluating software implementations. [126] used a focus group and a survey to (among other things) define the practice. They write, "[t]he questionnaire confirmed that respondents considered unit tests to be technical tests focused on the systemâĂŹs smallest units." As another example of work in this field, [150] discussed parameterizing unit tests and automation that can leverage such a notion.

At first glance, unit testing seems aligned with the DESC AI agent, particularly with the Simulation technique. For example, one could consider individual functions or behaviors as units of the bigger SBF* model. Simulation evaluates each independently although function evaluation involves behaviors and behavior simulation recursively simulations lower-level behaviors through subfunctions, breaking this boundary somewhat. However, a critical difference in purpose exists between DESC and unit testing: DESC exists to support the conceptual stage whereas unit testing operates over implementation details. Thus, the way one thinks of testing results may be different. For example, a flaw during conceptual design may lead to picking a different biological source case altogether whereas an implementation flaw is more likely to just imply fixing the immediate issue. Thus, DESC differentiates itself from unit testing.

Graph matching (or the problem of subgraph isomorphism) is another related field. The problem addressed by graph matching is to determine if a pattern in one graph exists in another graph. [54] defined the graph matching problem along with various considerations such as matching on structure versus semantics, and it presents a survey of work in this area. [131] discussed searching in trees and graphs, the latter at least one could view as applying the graph matching problem to search and thus an example of work in this field. The Structure-Mapping Engine (SME) [44] and analogical mapping in general can be viewed as an analogical reasoning lens on graph matching between two concepts, with SME taking a heuristic-based approach.

The Comparison technique of DESC is essentially posing the graph matching problem between two SBF* models of designs and reporting on differences when the two do not match perfectly. Compositional Mapping can be thought of as a meta-strategy that (via the user of Compositional Mapping defining the partitions) uses the rich SBF* semantics to decompose a large graph matching problem into a set of smaller ones, with each type of smaller problem addressable by its own locally optimized graph matching algorithm. Although this thesis does not delve into any potential overlap or contribution of graph matching and DESC (via Comparison) for scope reasons, it would be interesting as future work to explore that field (a) to determine if Compositional Mapping could usefully contribute ideas to the field, (b) to compare Compositional Mapping's performance with existing approaches, and/or (c) to get inspiration for partition mapping solvers to use within Compositional Mapping.

The goal behind **automated or automatic programming** is to automatically go from a specification to an implementation. [6], as part of its discussion, characterized an extended version of this problem as being of three parts: improving what can be automatically compiled and creating specification languages to leverage that, developing an interactive bridge to go from a desired specification language to what can be automatically compiled, and developing a desired specification language. [113] represents a work in this field. They developed a method for automated programming that can go from specification to the final software product.

A connection can be made between automated programming and DESC's ability to execute, so to speak, a design model for simulation. That is, an SBF* model of a design is a form of a high-level specification for that design (at least when discussing a prior design or a candidate design), and applying the Simulation technique to such a model takes this specification, applies a strict semantics to it (e.g., how to interpret an equation on an explanation), and generates inferred behaviors. That said, the output of DESC's Simulation technique is not an implementation of the design nor a step on the way towards it. Rather, the output could be instead viewed as an alternative specification that was generated by inspecting parts of the original specification (e.g., the `Start` state conditions, the transitions, and the transition explanations) and ignoring others (e.g., the non-`Start` state conditions). This alternative specification is compared against the original to check its internal consistency. That said, a potential connection between DESC and automated programming does raise an interesting idea for future work. Assume that more detailed models are created in later phases of the design process. Perhaps one form of evaluation could be to see if one could trace from the conceptual design (the candidate design) to these more detailed design models, which are themselves closer to the final implementation of the design. How one would do this is unclear and needs more thought, but being able (or not!) to explain or justify how one goes from an earlier version to a later version seems like a potential means of evaluating if the later versions fulfill the (perhaps implied) earlier specifications.

CHAPTER III

THE STRUCTURE-BEHAVIOR-FUNCTION STAR MODELING LANGUAGE

This chapter discusses the knowledge representation for design models given as input to DESC. It addresses the question: given that DESC seeks to evaluate the internal and external consistency of designs, what form should the models given as input take? Additionally, what form should the alternative models used to check external consistency take? These questions relate to the input model and alternative model portions of Figure 3.1, which depicts a high-level view of the DESC model evaluation process that serves the broader goal of incremental design revision. The answer to these questions: each design is articulated as an SBF* model. This chapter explains what such models are.

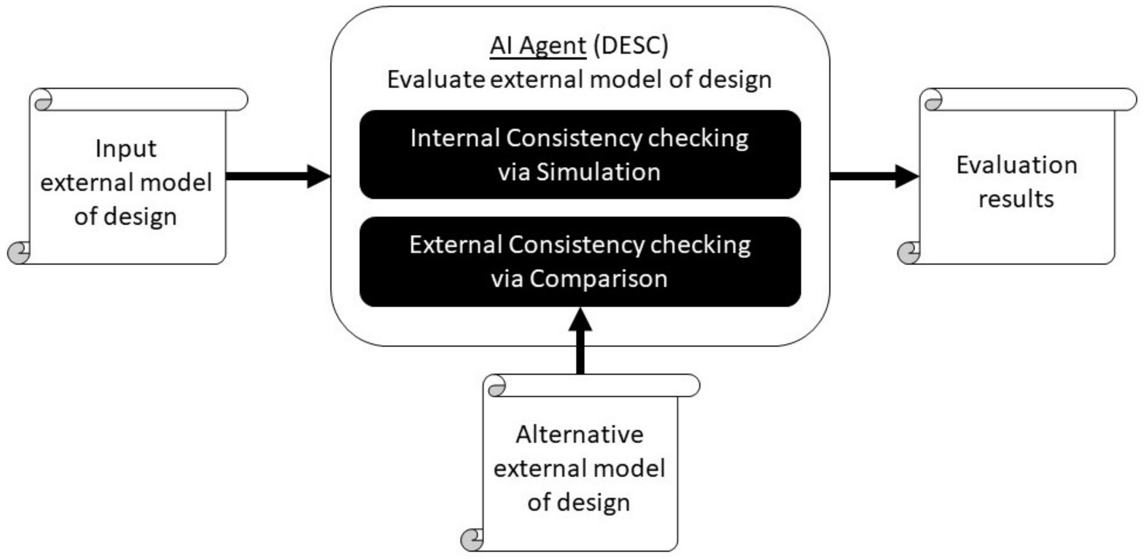

AI Agent (DESC)
Evaluate external model of design
Internal Consistency checking via Simulation
External Consistency checking via Comparison
Input external model of design
Evaluation results
Alternative external model of design

Figure 3.1: High-level process of design evaluation by DESC. This chapter addresses how input and alternative models are represented

The designs to be evaluated by the Design Evaluation through Simulation and Comparison (DESC) AI agent are represented in the Structure-Behavior-Function Star (SBF*) functional modeling language. SBF* is a version of the Structure-Behavior-Function modeling family of languages (e.g., [65]) that (in addition to being its own iteration of the language) has syntax and semantics to enable simulation capabilities. This chapter describes the SBF* language to help build the foundation for future discussion of the particular SBF* models used in this dissertation and of reasoning over models in this language. Extensions of SBF modeling enacted by SBF* are discussed at the end of the chapter.

SBF* acts as the representation for all models in all reasoning in this dissertation. That said, this dissertation does not actively use everything within SBF*. Thus, this chapter focuses only on those aspects that are utilized and, for brevity, ignores those that are not used or do not get used in a meaningful sense (e.g., connecting points on components).

Why use SBF* models? [112]'s description of conceptual design includes how practitioners should functionally decompose their design problem and then resolve these functions with design solutions. Thus, function is important in the conceptual stage of design, which is the stage targeted by this dissertation. SBF* models are a kind of functional model, which are models that give importance to function. Why use a functional model? Functional models (broadly speaking) and SBF* (specifically) align well with [112]'s characterization of how this design stage should proceed because of their emphasis on function. Function is explicitly represented in SBF*. Moreover, SBF* models describe how the system achieves its functions through behaviors. Moving back to the question of why use SBF* models, the family of SBF models has previously shown to be useful in computational reasoning tasks (e.g., [66]) and in supporting the field of biologically inspired design (e.g., [160]). While SBF* is not the same as the version of SBF in these prior works (itself a good thing because SBF* adds syntax and semantics for simulation that supports this dissertation), it is nevertheless in the same family and thus these works motivate its use in this dissertation's chosen biologically inspired design context.

All that said, it is important to note that SBF* models here are intended to be articulations of a design practitioner's internal (cognitive) understanding or vision of what is being modeled. Although the DESC AI agent is literally evaluating models and helping the practitioner to repair them, this is not the ultimate goal. Conceptually, DESC is really evaluating the practitioner's internal model of the design that is being articulated in the external model, and the goal is for both that internal model and the external SBF* model to be revised based on DESC's feedback.

A concrete example runs through this chapter to help explain the language: the Owl Wing model, which is one of the models developed for this dissertation (see Appendix D). This model was mentioned in Chapter 1 as part of the biologically inspired design example with the Train model. To briefly recap, the way an owl is able to fly quietly in the air inspired Shinkansen train employees to develop a small vortex generator to attach to the linkages (the pantographs) on their train. This helped to reduce the aerodynamic sound made by pantographs as the train moves. The small vortex generator mimics the mechanism by which owls fly quietly, which is depicted in the Owl Wing model. This model describes how the fimbriae on an owl's wings, which form the serrations on the edges of owl wings, enable it to fly while only making very low noise. Figure 3.2 shows another look at fimbriae in addition to the figures in Chapter 1. These serrations achieve this outcome by breaking up the air into microturbulences. These small vortices make only little noise.

3.1 High-level Description

The Structure-Behavior-Function modeling language is best thought of as a family of similar representations. The representation in this dissertation, SBF*, is also part of the SBF family. This version was initially co-developed in the Design and Intelligence Lab and has been since revised mostly to improve human readability and writability This revised version is presented herein, and the complete formal definition of the language is in Appendix A.

The Structure-Behavior-Function Star (SBF*) modeling language describes physical

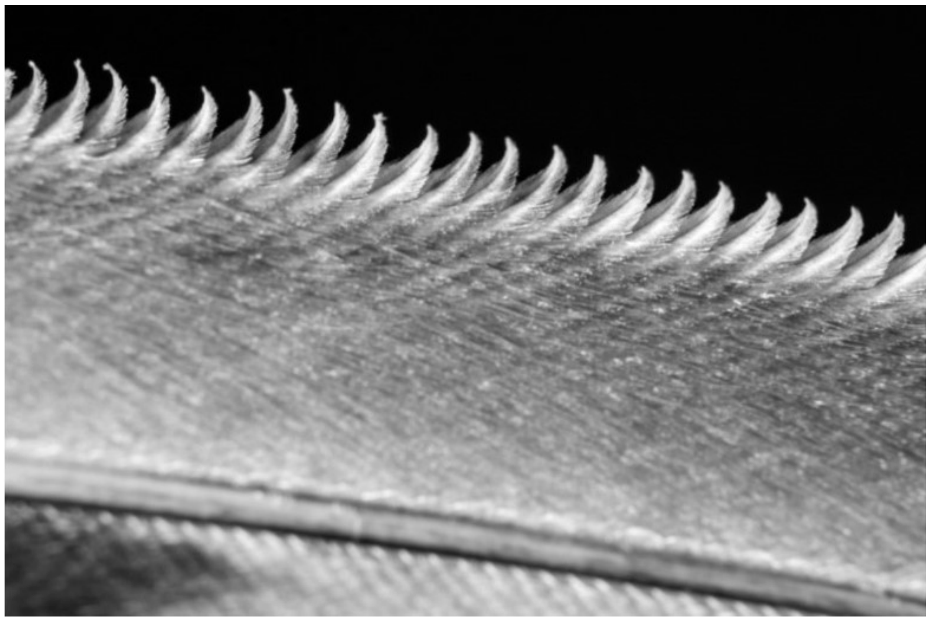


Figure 3.2: Fimbriae on the leading edge of barn owl primary feathers. Adapted from [149]

systems (as situated in this dissertation), such as technological devices or biological organisms. Conceptually, it breaks a system down into three submodels that align with the three parts of the language's name: the system's structure, functions, and behaviors. We can also say that these submodels describe the system's parts, objectives, and mechanisms, respectively.

The behaviors of a system describe how it achieves its functions. A behavior can point to a subfunction that helps in the process. In doing so, this forms a function-subfunction relation, with the superfunction being the function achieved by the behavior in question and the subfunction being the function that the behavior points to. If one were to combine the total set of function-subfunction relations in a model, this forms the functional decomposition of the model, displaying how all of its objectives (functions) relate to each other hierarchically.

The functional decomposition of the Owl Wing model is visualized in Figure 3.3. White boxes are functions and the black circles represent associated behaviors. One can see that a function-subfunction relationship between OwlFliesSilently and FimbriaeBreakDownAir is mediated by the behavior of the former. High level descriptions of the functions are provided when discussing the functions submodel below.

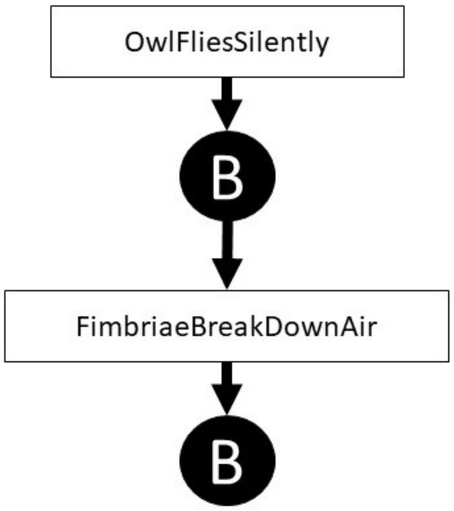


Figure 3.3: Functional decomposition of the Owl Wing model with associated behaviors

3.2 *The Structure Submodel*

The structure submodel describes the parts of the system and how they connect to each other. The structure submodel consists of components (Owl and Air) and their attributes (Moving, NoiseCreated, MovingPastOwl, and TurbulenceSize). A component has a name, possibly a textual description, and zero to many attributes. An attribute has a name, and it has a type of either Quantitative (meaning its values are numerical), Qualitative (its values come from a defined quantity space), or Descriptive (its values are any text string). An attribute may also have a text description, which should define the quantity space of that attribute if it is of the Qualitative type but in other contexts is only for human readers. A quantity space is an ordered sequence of terms that each represent a quantity on a spectrum. See the section in Chapter 4 on DESC's Simulation technique for particulars of quantity spaces in SBF*.

For an example of these modeling elements, consider the model snippet in Figure 3.4 from the Owl Wing model in the text syntax for SBF*. It shows the component `Owl` and its two attributes `Moving` and `NoiseCreated`, both of `Qualitative` type with their own quantity spaces. (See note below about the connecting point.)

```
Owl is a component
  It has a connecting point named C1
  It has an attribute Moving of type Qualitative, described as
     "False, True"
  It has an attribute NoiseCreated of type Qualitative,
     described as "Zero, Very_Low, Low, Medium, High"
```

Figure 3.4: Structure submodel snippet of a component from the Owl Wing model

This submodel also contains connections. A connection has a name and possibly a textual description. A connection declares that two components are connected to each other by some relation and at connecting points of those components. Although models in this dissertation use connecting points to connect two components, connecting points are all given the same placeholder name (`C1`) for simplicity and so will not be mentioned further.

For an example of a connection, consider the model snippet in Figure 3.5 from the Owl Wing model. This depicts a connection named `WingIsPartOfAnOwl`, which describes how a `Wing` and `Owl` are connected through a `PartOf` mechanism.

```
Wing.C1 is PartOf Owl.C1 is a connection named
   WingIsPartOfAnOwl
```

Figure 3.5: Structure submodel snippet of a connection from the Owl Wing model

Figure 3.6 depicts diagrammatically the complete structure submodel for the Owl Wing model. Rounded boxes represent the components, and they contain the component's attributes. Edges represent connections, and they are annotated with connecting points (not covered here), the name of the connection, and the connection's mechanism. The structure submodel of this model depicts an Owl with wings, and wings with serrations. Alongside this is the component of `Air`, which (although not depicted as a connection) the owl moves through.

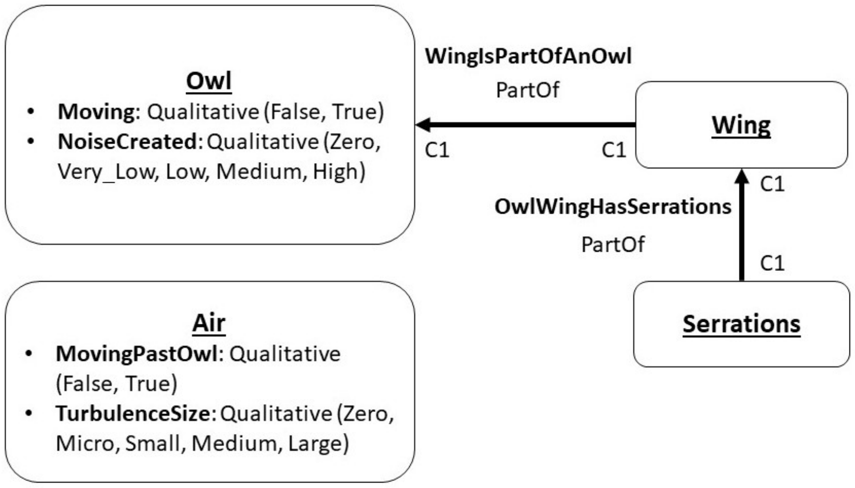

Owl
• Moving: Qualitative (False, True)
• NoiseCreated: Qualitative (Zero, Very_Low, Low, Medium, High)
Air
• MovingPastOwl: Qualitative (False, True)
• TurbulenceSize: Qualitative (Zero, Micro, Small, Medium, Large)
WingIsPartOfAnOwl
PartOf
C1
C1
Wing
C1
OwlWingHasSerrations
PartOf
C1
Serrations

Figure 3.6: Structure submodel of the Owl Wing model

3.3 The Functions

This submodel captures a set of functions, each describing either the intended or perceived purpose of the system or some subsystem. A function consists of a name, an optional textual description, an optional verb that abstracts what it does to a single word, and a pointer to the behavior that achieves the function. A function also consists of a set of provides conditions[1], which stipulate values of component attributes (each written as `Component.Attribute`) that should be true at the completion of the function. A function also contains requires conditions, which stipulate values of component attributes that should be true at the beginning of the function. However, this feature of SBF* modeling is never

[1]Technically, a function contains a single provides condition that itself contains one to many primitive conditions. This dissertation document simplifies the language in SBF* modeling for conciseness by essentially merging the two layers together and referring to each primitive condition as an individual provides condition amongst a set of provides conditions. No precision is lost by doing so, and all conditions in functions and states in SBF* models follow a similar convention in this dissertation document.

reasoned with in this dissertation and so is ignored.

Consider the function `OwlFliesSilently` from the Owl Wing model, depicted in Figure 3.7. This function is achieved by the behavior named `OwlFliesSilentyBehavior`. This function ascribes flying silently as the purpose of the owl, which makes sense in the context that the system inspired a means by which engineers reduced the noise made by their train. The provides condition of this function says that the amount of `NoiseCreated` by the `Owl` will be `Very_Low`, fitting with the notion of flying silently. (Admittedly, that it produces `some` noise is dissonant with the function name.)

```
OwlFliesSilently is a function
  It requires nothing
  It provides: (
    Owl.NoiseCreated = "Very_Low"
  )
  It is achieved by the behavior OwlFliesSilentlyBehavior
```

Figure 3.7: Function from the Owl Wing model

Figure 3.8 presents a more detailed version of Figure 3.3 that includes both the provides conditions for the two functions in the Owl Wing model and the names of the behaviors that achieve the functions. White boxes represent functions and rounded black boxes represent the associated behaviors. The top-level function `OwlFliesSilently` defines making very low noise as a function of the owl. Inspecting the behavior associated with this function, one sees that this function is in the context of the owl moving and air moving past the owl. This function has a subfunction, `FimbriaeBreakDownAir`. This function says that the owl also has the function of creating micro-sized turbulence in the air. Inspecting its associated behavior, one sees that the modeler decided that the `PartOf` connection between the serrations and owl's wing is responsible for moving the turbulence size towards the `Micro` value when air is moving past the owl.

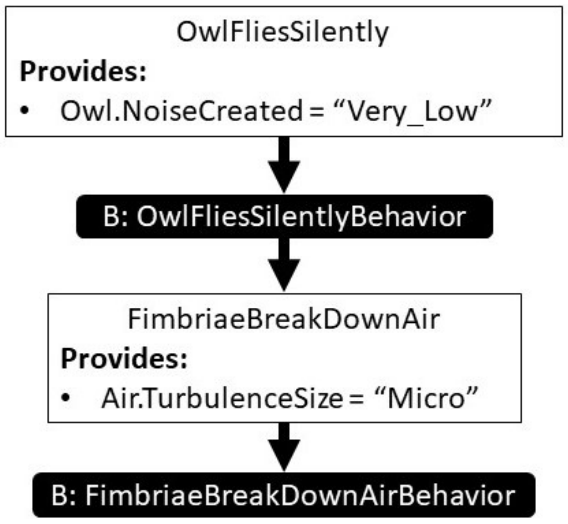

OwlFliesSilently
Provides:
• Owl.NoiseCreated = "Very_Low"
B: OwlFliesSilentlyBehavior
FimbriaeBreakDownAir
Provides:
• Air.TurbulenceSize = "Micro"
B: FimbriaeBreakDownAirBehavior

Figure 3.8: More detailed functional decomposition of the Owl Wing model

3.4 The Behaviors

This submodel captures a set of behaviors, each describing the process by which the system fulfills a given function. A behavior has a name, an optional textual description, and zero to many states and transitions. Consider the behavior in Figure 3.9 from the Owl Wing model, `OwlFliesSilentlyBehavior`. This behavior describes (a) the owl initially still then (b) the owl deciding to move, which causes air to move past it. Next, (c) the owl's fimbriae break down the air into micro turbulences, which create very low noise.

Figure 3.10 depicts this behavior diagrammatically to aid in understanding. Boxes represent states and arrows represent transitions between the states. The behavior should be read as beginning from `StartState` and completing at `StopState` as it does in the textual representation.

For completeness, below also see the diagrammatic (Figure 3.11) and textual (Figure 3.12) versions of the `FimbriaeBreakDownAirBehavior` behavior. Like before, the boxes in the figure represent states and the arrows between the boxes represents the transition.

A behavior has zero to many states, such as `StartState` in the example behavior. Conceptually, a state describes the values of component attributes at a moment in time. A

```
OwlFliesSilentlyBehavior is a behavior

 StartState is the start state
  It has the condition: (
   Owl.Moving = "False"
   and Air.MovingPastOwl = "False"
   and Owl.NoiseCreated = "Zero"
  )

 T1 is a transition from StartState to State2
  It has the explanation: (
   the external stimulus named OwlDecidesToFly, described as "
       eq: Owl.Moving is equal to Owl.Moving.True"
   and the equation AirMovesPastOwlEQ, described as "eq: Air.
       MovingPastOwl is directly proportional to Owl.Moving:
       After - Owl.Moving:Before"
  )

 State2 is a state
  It has the condition: (
   Owl.Moving = "True"
   and Air.MovingPastOwl = "True"
  )

 T2 is a transition from State2 to StopState
  It has the explanation: (
   the function FimbriaeBreakDownAir
   and the equation TurbulenceCausesNoise, described as "eq:
       Owl.NoiseCreated is equal to Air.TurbulenceSize:After"
  )

 StopState is the stop state
  It has the condition: (
   Owl.NoiseCreated = "Very_Low"
  )
```

Figure 3.9: A behavior from the Owl Wing model

state consists of a name, an optional textual description, and optional flags to declare if it is a start or stop state. A start state (such as `StartState` in the example) is a state that begins the sequence of states in the behavior. The stop state (`StopState` in the example) is a state that ends it. The absence of such flags implicitly declares that the state is intermediate

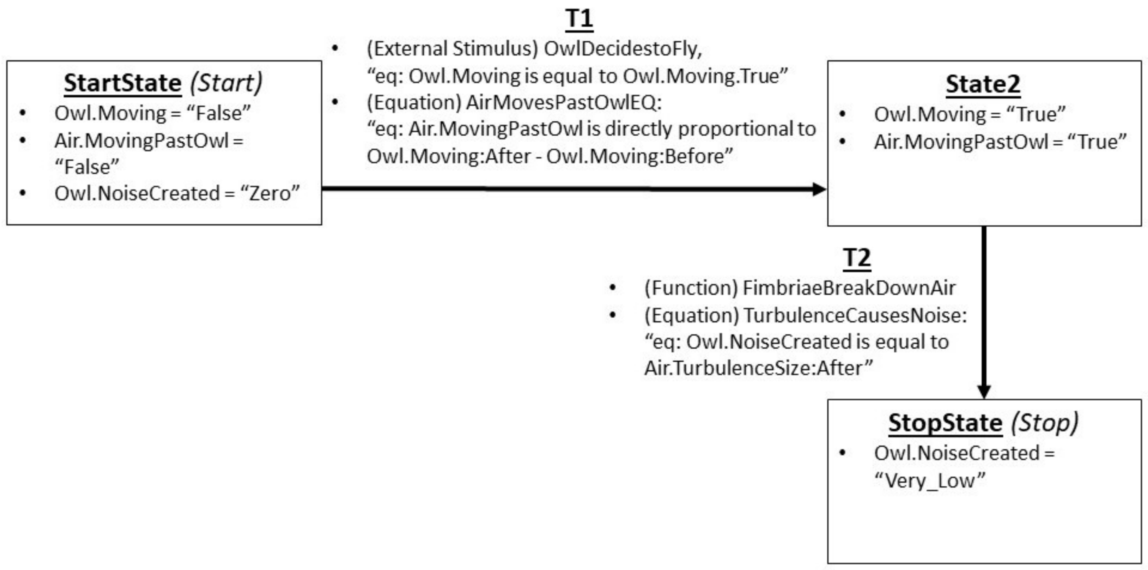

T1
(External Stimulus) OwlDecidestoFly, "eq: Owl.Moving is equal to Owl.Moving.True"
(Equation) AirMovesPastOwlEQ: "eq: Air.MovingPastOwl is directly proportional to Owl.Moving:After - Owl.Moving:Before"
StartState (Start)
Owl.Moving = "False"
Air.MovingPastOwl = "False"
Owl.NoiseCreated = "Zero"
State2
Owl.Moving = "True"
Air.MovingPastOwl = "True"
T2
(Function) FimbriaeBreakDownAir
(Equation) TurbulenceCausesNoise: "eq: Owl.NoiseCreated is equal to Air.TurbulenceSize:After"
StopState (Stop)
Owl.NoiseCreated = "Very_Low"

Figure 3.10: Diagram of the `OwlFliesSilently` behavior in the Owl Wing model

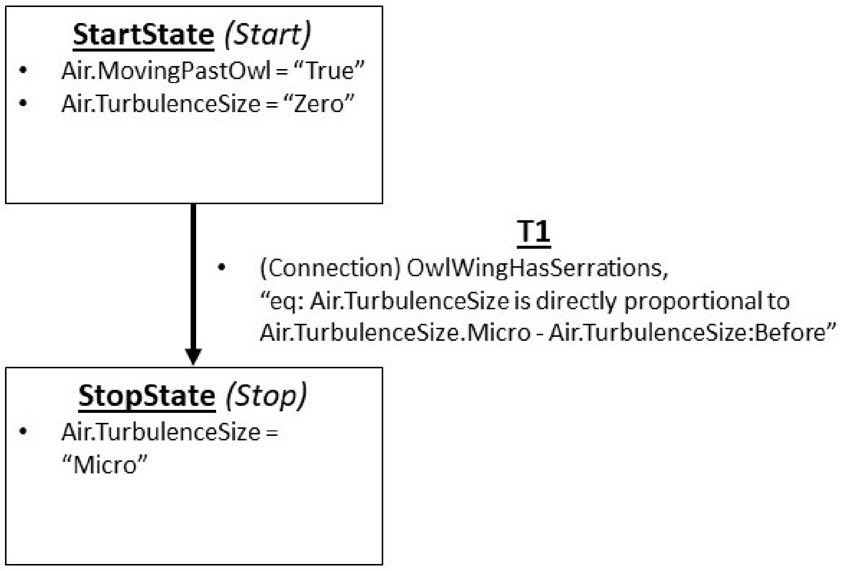

StartState (Start)
Air.MovingPastOwl = "True"
Air.TurbulenceSize = "Zero"
T1
(Connection) OwlWingHasSerrations, "eq: Air.TurbulenceSize is directly proportional to Air.TurbulenceSize.Micro - Air.TurbulenceSize:Before"
StopState (Stop)
Air.TurbulenceSize = "Micro"

Figure 3.11: Diagram of the `FimbriaeBreakDownAir` behavior in the Owl Wing model

(`State2` in the example), which means it occurs somewhere in the middle of the sequence of states.

For simplicity, this dissertation assumes that behaviors consist of a linear sequence of states with exactly one start state, exactly one stop state, at most one incoming transition per

```
FimbriaeBreakDownAirBehavior is a behavior

 StartState is the start state
  It has the condition: (
   Air.MovingPastOwl = "True"
   and Air.TurbulenceSize = "Zero"
  )

 T1 is a transition from StartState to StopState
  It has the explanation: (
   the connection OwlWingHasSerrations, described as "
      eq: Air.TurbulenceSize is directly proportional
      to Air.TurbulenceSize.Micro - Air.TurbulenceSize:
      Before"
  )

 StopState is the stop state
  It has the condition: (
   Air.TurbulenceSize = "Micro"
  )
```

Figure 3.12: Another behavior from the Owl Wing model

state, and at most one outgoing transition per state. See discussion in Chapter 7 about the extent to which DESC might handle behaviors that violate this assumption (e.g., non-linear behaviors).

A state also consists of zero to many conditions. A condition defines the value of a component's attribute (written as `Component.Attribute`) at the moment of time represented by the state it is in, and the set of them in a state forms a conjunction. That is, all of them are true. For example, in `StartState`, we see that the `Moving` attribute of `Owl` is `False`, meaning that the owl begins this behavior not moving.

Additionally, when reading a behavior in an SBF* model, an important modeling assumption should be noted. If a component attribute in a previous state does not appear in a subsequent state, this implies that the component attribute's value holds from what it was defined as most recently. This assumption resolves the Frame Problem [130] for behaviors such that we assume only those component attributes explicitly mentioned have changed between states. (That said, one may still enumerate component attributes with values that remain the same relative to their previous appearance if one so chooses.) Beyond helping the readability of SBF* models by simplifying them, this assumption becomes important when simulating models, where it is formalized as Implicit Value Forwarding (see Chapter 4).

A behavior also has zero to many transitions. Whereas a state describes a moment in time, a transition describes how or why the system moves from one state to the next. A transition has a name (technically, this is optional); it has an optional textual description; and it points to the state preceding it and to the state following it. For example, the transition named `T1` in the example `OwlFliesSilentlyBehavior` behavior has no textual description and says that it goes from the state `StartState` to the state `State2`.

A transition also consists of zero to many explanations[2]. Explanations come in many types, but they all serve the same purpose: describing how or why one or more component attribute values change in the transition between states. For example, consider the explanations in transition `T2` in the example `OwlFliesSilentlyBehavior` behavior. The first explanation is of type function, meaning that the behavior executes the subfunction `FimbriaeBreakDownAir` of the system when it moves from state `State2` to `StopState`. Conceptually, this subfunction refers to how the serrations in owl wings produce microturbulences in air. In practice, we see that this results in the `TurbulenceSize` of `Air` being set to `Micro`. The second explanation is called `TurbulenceCausesNoise` and is of type equation. This explanation relates the amount of noise created by the owl to the size of turbulences that it creates.

[2]Explanations are also simplified linguistically in this dissertation like conditions are. A transition technically contains an optional explanation which itself contains one to many primitive explanations. Here, a primitive explanation is instead referred to as an explanation in a set of explanations.

*3.5 Extensions of Structure-Behavior-Function Modeling in SBF**

The SBF* language is, intentionally or not, conceptually similar to the version of SBF presented in [65]. This dissertation extends the SBF language (mostly considering how it differs from that version, but also in a general sense) in a few ways that, in addition to generally making the language more precisely defined, add syntax and semantics to facilitate automated simulation, addressing Hypothesis 2.2. This section surveys major ways that SBF* extends SBF. Note that some aspects, such as quantity spaces, are tightly connected to the Simulation technique of DESC and are only touched upon in this section. Chapter 4 describes their use in more detail.

SBF* extends SBF modeling in several ways. Table 3.1 summarizes the extensions and is followed by a more thorough walkthrough.

Table 3.1: SBF* extensions on SBF

Model Element	**Description**
Component attribute	Defined `Quantitative`, `Qualitative`, and `Descriptive` types
Component attribute	Quantity space syntax and semantics for `Qualitative` type
Behavior explanation	Attribute value semantics for function-type explanations
Behavior explanation	Formal Equation syntax and semantics for non-function-type explanations
Behavior	Formalizing meaning of missing attribute values in states

In the structure submodel, SBF* extends the language in two ways. First, all attributes are typed as either `Quantitative`, `Qualitative`, or `Descriptive`, and each of these types connotes different expectations about the values assignable to that attribute. `Quantitative` attribute values should be numbers; `Qualitative` attribute values should correspond to words or phrases that exist within a defined quantity space; `Descriptive` attribute values should be textual. Second, if an attribute is `Qualitative`, its textual description carries a particular meaning. The attribute's description should hold its quantity space, and this quantity space is used in the Simulation technique of DESC to reason about the values of the attribute.

In a behavior, SBF* extends the language in three ways that are both related to transition explanations. First, SBF* precisely defines the semantics of a function explanation relative to the attribute value state changes it causes (attributes in the function's provides condition plus any attributes whose values changed). Second, SBF* defines a syntax and semantics for quantitative, qualitative, and descriptive equations in all other explanation types, overloading the textual description for those types to reason about equations when the particular trigger is present to do so. Function explanation and equation reasoning are covered in detail in Chapter 4. Third, SBF* formalizes the modeling assumption that unspecified attribute values in successive states implies that the most recently defined value holds.

3.6 Summary

To summarize, a Structure-Behavior-Function Star (SBF*) model of a system describes its structure, functions, and behaviors. The structure submodel of a system describes its component parts, their attributes, and any connections that exist between those components. The functions of a system describe either its intended or perceived purposes. Each function is achieved in the system by a behavior. Behaviors are presented as state-transition diagrams. They depict how the component attribute values of the system change through time along with how and why this change occurs.

It is important to note the interconnectivity of the model elements within SBF*. First, components and attributes that appear in the structure submodel can also reappear within the behaviors and functions. This conceptually links all three submodels in the sense that they are all talking about the same physical parts of the system when the same component and attribute names are used. Second, functions can appear as subfunction explanations in transitions in behaviors. This allows one to compose a functional decomposition of the system through these subfunction explanations and, if the tree goes deeper than one level, mediating behaviors. In the Owl Wing example, the behavior of the function `OwlFliesSilently` makes a subfunction call to `FimbriaeBreakDownAir`. Thus, `FimbriaeBreakDownAir` is a subfunction of `OwlFliesSilently`.

This chapter has shown examples from the Owl Wing model in both the textual (formal) representation of SBF* models and in more visual diagrammatic form. The implementation of DESC created for this dissertation ingests models in the textual representation form. The full versions of all SBF* models used in this dissertation in their textual form can be found in Appendix D.

CHAPTER IV

DESIGN EVALUATION THROUGH SIMULATION AND COMPARISON

This chapter discusses the two techniques, Simulation and Comparison, enacted by the AI agent called DESC, for Design Evaluation by Simulation and Comparison. In so doing, this chapter provides a detailed answer to the question: how might an AI agent support incremental design revision? The answer: it may do so by (mostly) automatically checking for internal consistency and external consistency issues. Figure 4.1 depicts a high-level process for how DESC, via its two techniques, will take an external design model as input, reason about internal consistency (via Simulation) and external consistency (via Comparison) and produce feedback for a practitioner as output.

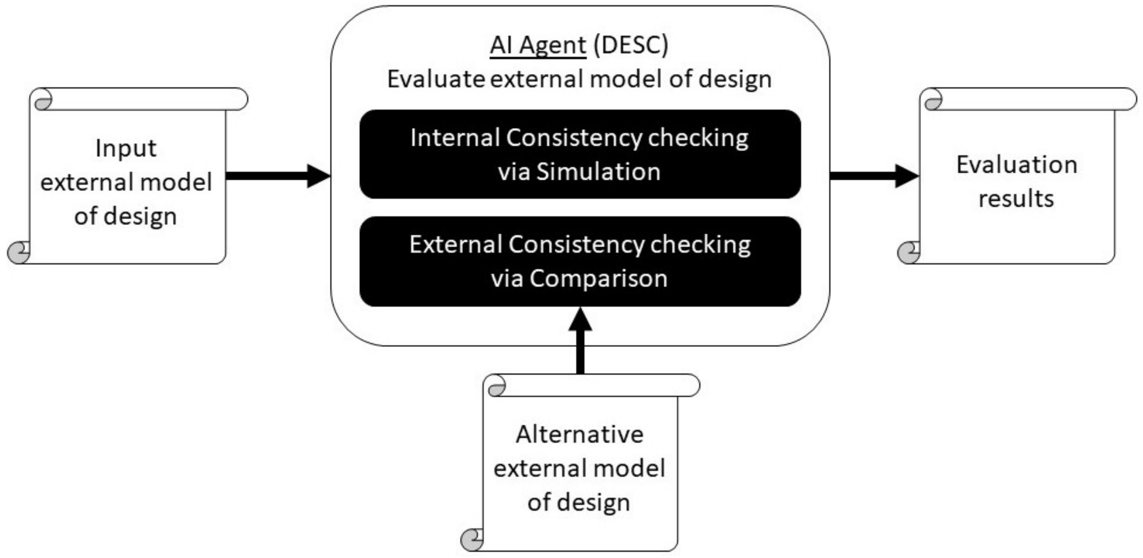

AI Agent (DESC)
Evaluate external model of design
Internal Consistency checking via Simulation
External Consistency checking via Comparison
Input external model of design
Evaluation results
Alternative external model of design

Figure 4.1: High-level process of design evaluation by DESC. This chapter addresses how the internal consistency checking (Simulation) and external consistency checking (Comparison) techniques work

This chapter presents detailed descriptions of the Simulation and Comparison techniques. Further, this chapter also describes this dissertation's novel analogical mapping algorithm Compositional Mapping that is used within the Comparison technique as well the Uniform Mapping algorithm that is intended to represent a more conventional approach. This chapter is written with the assumption that readers are already familiar with the Structure-Behavior-Function Star (SBF*) functional modeling language, covered in Chapter 3. However the syntax and semantics of equations in transition explanations are discussed exclusively in this chapter because they are tightly coupled to Simulation.

Note that process diagrams, pseudocode, narrative descriptions, etc. present an idealized and sometimes simplified view of the actual processes and representations developed to implement DESC's techniques. They are intended as detailed technical descriptions and capture the essential ideas of DESC, but the precise implementation created for this dissertation may not align exactly[1], so they (the pseudocode especially) should not be interpreted as actual code or computer architecture except when specifically discussing the technical architecture.

This chapter is broken down into the following sections. First, it provides a high-level description of DESC. Then each of the two parts of DESC: Simulation then Comparison (including its two mapping algorithms) are given a section for detailed discussion.

4.1 Overview

The Design Evaluation through Simulation and Comparison (DESC) AI agent addresses how an AI agent will check for internal and external consistency to support incremental design revision. This encapsulates the Hypotheses for Research Questions 2 and 3 in this dissertation. Recall:

- **Research Question 2.** How might an AI Agent check a design model for internal

[1]For example: the current implementation of the Simulation technique has complexity to enable multiple trajectories due to ambiguous qualitative equations. This was never fully explored and so was scoped out of this dissertation, and thus it is entirely ignored in the following descriptions.

consistency?

Hypothesis 2.1. It may do so by simulating the model and using the results to check the process and functional claims made by the model. The Simulation technique of DESC illustrates this hypothesis.

Hypothesis 2.2. The SBF* functional modeling language provides the syntax and semantics needed for automated simulation.

- **Research Question 3.** How might an AI agent check a design model for external consistency?

  Hypothesis 3. It may do so by analogically comparing a given model against another model of the same design, where differences represent potential issues. The Comparison technique of DESC illustrates this hypothesis.

Accordingly, DESC presents two techniques for evaluating articulations of designs in a functional model representation schema. DESC assumes these representations are in the Structure-Behavior-Function Star (SBF*) modeling language. These models are externalizations of a reasoner's internal, perhaps cognitive (if human), understanding or envisioning of the designs. Any issues detected in the models may thus reflect issues in the reasoner's internal understanding or envisioning of the design that need to be repaired or rethought.

DESC is a package for two techniques. The techniques are independent of each other in their operation, but they are conceptually complementary with different purposes and approaches to evaluation. One way to think about their purposes is along the lines of internal and external consistency checking.

Internal consistency, an inward-looking form of evaluation, is meant as the extent to which aspects of the model agree with each other. For instance, each function has a set of provides conditions that it claims will be true at its completion. Internal consistency might ask, "does the behavior associated with this function produce output that is consistent with these provides conditions?" If this is not the case, there exists an internal consistency issue in the model. The Simulation technique of DESC (Section 4.4) evaluates the internal consistency of a model.

External consistency, an outward-looking form of evaluation, is meant as the extent to which the model accurately represents the modeled concept as it is in the world, relative to the purpose of the model. For candidate designs, which have no real-world counterpart, external consistency instead is meant as the extent to which the model aligns with those of the practitioner's teammates or any external representation of the design (assuming either exist). For instance, external consistency might ask, "does the practitioner's model have the same set of components in its structure submodel as a teammate's model?" If it does not, this difference indicates a potential external consistency issue in one or both of the two models that should be further investigated. The assumption of which model is likely at fault depends on if either model is authoritative relative to the other. The Comparison technique of DESC (Section 4.5) evaluates the external consistency of a model.

Another lens with which to view DESC's techniques is in terms of verification and validation. The Simulation technique verifies and validates models given to it. The meaning of verification here is adapted from Boehm's [12] informal definition in the context of software: "Am I building the product right?" In the context of designs, this dissertation operationalizes verification as evaluating the design in terms of the syntax and semantics (the rules and the meaning, respectively) of the representational language in which it is expressed. Specifically, the Simulation technique directly targets the semantics. Simulation does so by checking to what extent a model's behaviors are internally coherent.

Simulation also validates designs. The meaning of validation here is adapted from Boehm's [12] informal definition in the context of software: "Am I building the right product?" Simulation does validation by checking to what extent the behaviors produce the output expected by their associated functions. In other words, does the design actually achieve its functions? Simulation thus achieves verification and validation by simulating the behaviors of the given model and comparing the results of that simulation (state conditions) against conditions given by the modeler in the behavior and in the functions' requires conditions.

The Comparison technique also validates designs, where validation is operationalized as how accurately (relative to the purpose of the model) the design model depicts the design being articulated. This applies to design concepts that already exist in the world or for which there somehow exist multiple models (e.g., a candidate design that has been independently modeled by two or more teammates). Comparison achieves validation by comparing a given design model against another model of the same design concept, which provides a different perspective on the concept. Any differences detected represent potential issues to be further investigated.

4.2 Running Example: Medical Patch

To ground discussion on DESC, this chapter uses a model of a (biologically inspired) technology design as a running example: the Medical Patch model. [101] describes the lab of Dr. Jeffrey Karp, who's group works in the field of invasive surgery. As McKeag colorfully puts it, "Jeffrey Karp has been on a quest to find a better way to stick material onto things that are wet and gooey and move around" [101, p. 20]. Karp's team of engineers searches for natural systems that will help them find the best wound closing and tissue repairing solutions.

The Medical Patch model is derived from one such solution briefly described in [101]: a medical adhesive path whose adherence mechanism is inspired by the spiny headed worm. This worm adheres to its host by inserting a part of its body into the host and expanding it, using friction to hold itself in place. This "friction fitting" is depicted in Figure 4.2.

Karp and team developed a medical patch by inspiration to this worm. An array of microneedles in a patch are inserted. Once done, they their tips swell to create fiction and hold the patch in place. This is similar to how the worms hang on. This bioinspired medical

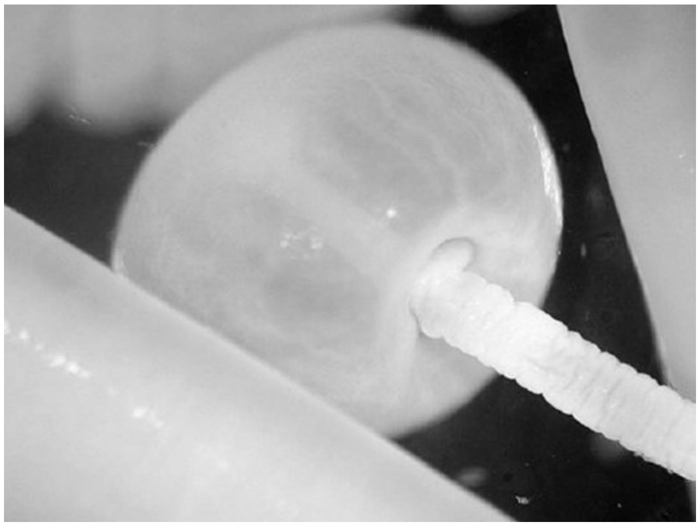


Figure 4.2: The Spiny Headed Worm expands part of its body to adhere by friction. Adapted from [101]

patch's adhesion strength is 3.5 times greater than that of staples that are typically used in skin grafting. It has other benefits too, such as a reduced infection risk compared to staples and the ability to deliver drugs.

The Medical Patch model attempts to describe this solution. It describes, in its top-level behavior, a patch being inserted, the conical tips of the microneedle array in the patch expanding, and the patch resisting an applied pull force because of the friction created by the expanded tips. The textual form of this model can be viewed in Appendix D under the same name. The below figures are diagrammatic recreations to aid in understanding.

Figure 4.3 depicts the structure submodel of the Medical Patch. This submodel contains 5 components (rounded boxes), 6 connections (edges), and 8 attributes (bullets inside the components). The attributes are a mixture of Quantitative and Qualitative types. The quantity spaces, from lowest to highest quantity, for the Qualitative-type attributes are given in parentheses after the type.

Figure 4.4 depicts the functional decomposition of the Medical Patch model. There are 3 functions (white boxes), each with an associated behavior (black boxes). The function `PatchResistsPullout` is a superfunction to `TipsSwell` and

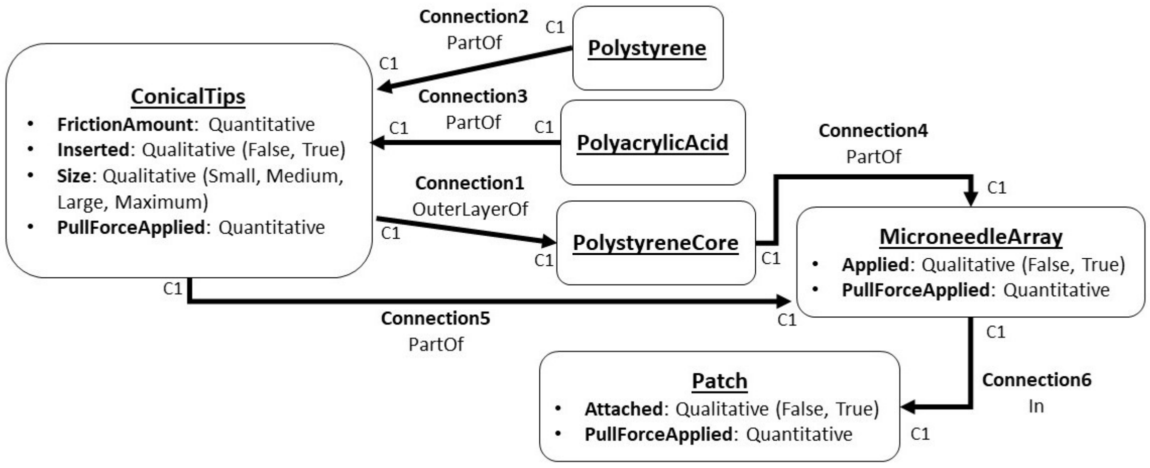

Connection2
PartOf
C1
Polystyrene
C1
ConicalTips
• FrictionAmount: Quantitative
• Inserted: Qualitative (False, True)
• Size: Qualitative (Small, Medium, Large, Maximum)
• PullForceApplied: Quantitative
Connection3
PartOf
C1
C1
PolyacrylicAcid
Connection4
PartOf
C1
Connection1
OuterLayerOf
C1
C1
PolystyreneCore
C1
MicroneedleArray
• Applied: Qualitative (False, True)
• PullForceApplied: Quantitative
C1
Connection5
PartOf
C1
C1
Patch
• Attached: Qualitative (False, True)
• PullForceApplied: Quantitative
Connection6
In
C1

Figure 4.3: Structure submodel for the Medical Patch model

`TipsResistPulloutBehavior`. Within each function box is that function's set of provides conditions (bullets), each using the `<Component>.<Attribute> = <Value>` construction. Each of these functions has only a single provides condition.

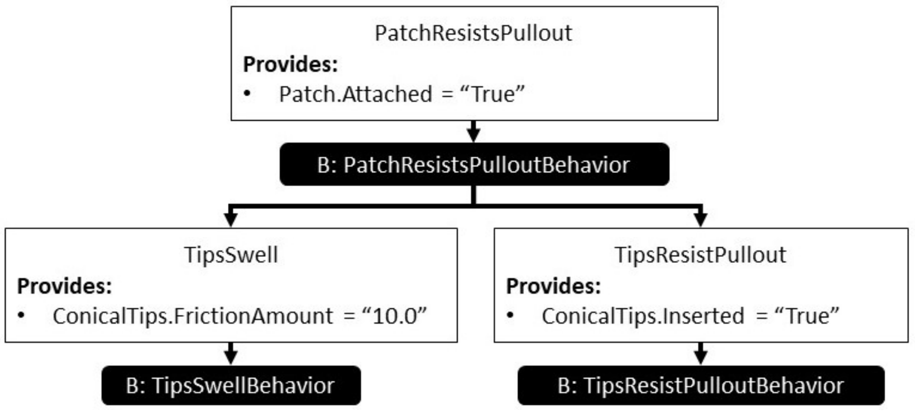

PatchResistsPullout
Provides:
• Patch.Attached = "True"
B: PatchResistsPulloutBehavior
TipsSwell
Provides:
• ConicalTips.FrictionAmount = "10.0"
B: TipsSwellBehavior
TipsResistPullout
Provides:
• ConicalTips.Inserted = "True"
B: TipsResistPulloutBehavior

Figure 4.4: Functional decomposition for the Medical Patch model

Figure 4.5 depicts the `PatchResistsPullout` behavior. The behavior describes how the patch becomes attached, the tips swell, some force pulls on the patch, but it stays attached because the tips resist the patch being pulled out. This behavior contains 5 states (boxes), 4 transitions (edges), 13 state conditions (bullets in states), and 6 transition explanations (bullets next to transitions). Note that equations are not shown in this behavior diagram for visual simplicity.

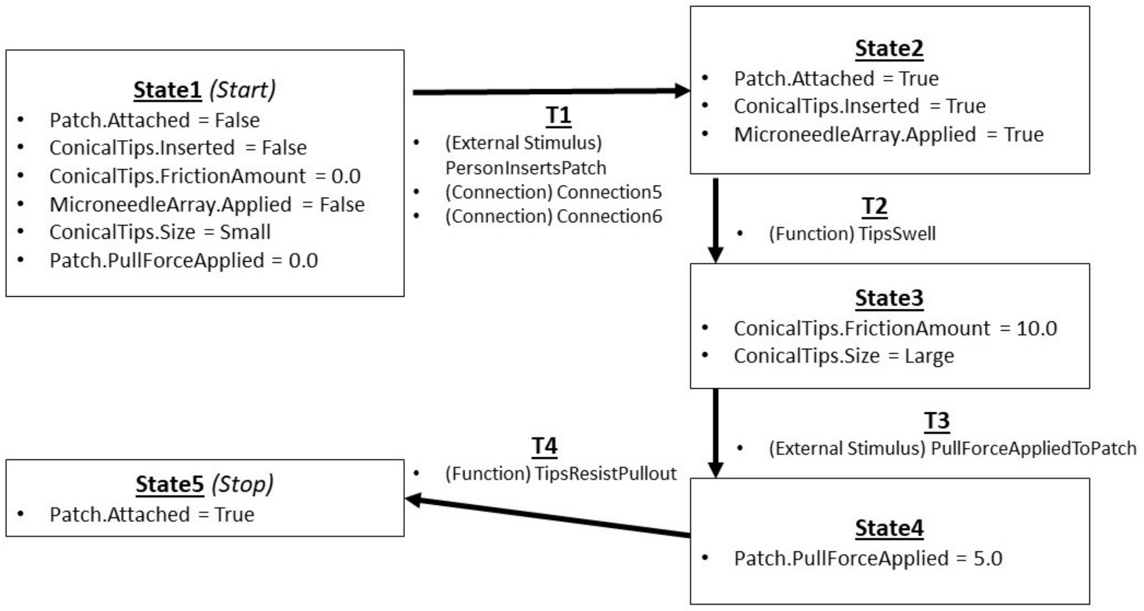

State1 (Start)
Patch.Attached = False
ConicalTips.Inserted = False
ConicalTips.FrictionAmount = 0.0
MicroneedleArray.Applied = False
ConicalTips.Size = Small
Patch.PullForceApplied = 0.0
T1
(External Stimulus) PersonInsertsPatch
(Connection) Connection5
(Connection) Connection6
State2
Patch.Attached = True
ConicalTips.Inserted = True
MicroneedleArray.Applied = True
T2
(Function) TipsSwell
State3
ConicalTips.FrictionAmount = 10.0
ConicalTips.Size = Large
T3
(External Stimulus) PullForceAppliedToPatch
State4
Patch.PullForceApplied = 5.0
T4
(Function) TipsResistPullout
State5 (Stop)
Patch.Attached = True

Figure 4.5: `PatchResistsPullout` behavior for the Medical Patch model. Equations are not shown for simplicity

4.3 Technical Architecture

An operational computer program was implemented in Java for each of the two techniques of DESC. The end of each section on Simulation and Comparison briefly discusses the final version of that implementation's technical architecture. Although each share some common architectural elements, the technique implementations are largely separate from each other (as they are at a theoretical level).

The architecture is presented at a high level of detail such that small technical things, such as the specific data structures used and utility algorithms, are ignored in favor of the important, logical content and processes. Additional implementation details, such as the harnesses created to run the Computational Experimentation (see Chapter 5), are also scoped out unless necessary.

4.4 Simulation Technique

Simulation is one of DESC's two techniques. It addresses the following research question, and it directly addresses Hypothesis 2.1. Additionally, through motivating and leveraging features of SBF* (see Chapter 3) to facilitate its automated simulation-based reasoning strategy, it thus also indirectly addresses Hypothesis 2.2.

- **Research Question 2.** How might an AI Agent check a design model for internal consistency?

  Hypothesis 2.1. It may do so by simulating the model and using the results to check the process and functional claims made by the model. The Simulation technique of DESC illustrates this hypothesis.

  Hypothesis 2.2. The SBF* functional modeling language provides the syntax and semantics needed for automated simulation.

In essence, Simulation hypothesizes that one can evaluate the internal consistency of a functional model by simulating it and then reflecting on the simulation results to check claims made by the representation. This technique assumes that the idea of simulation makes sense given the knowledge representation. In this dissertation, Simulation is thus grounded in such a representation: the SBF* functional modeling language. SBF* models have behavioral descriptions that describe processes undergone by the concept being modeled. The descriptions of Simulation in the remainder of this chapter are also grounded in this language, but the broader theory of evaluation through simulation can transcend this single language, as evidenced, in the general sense, by related works in Chapter 2 that also use simulation for evaluation.

The following sections go into detail about how Simulation reasons about SBF* models and evaluates them. The high level process of Simulation is depicted in Figure 4.6. Given an input SBF* model, (step 1) Simulation will simulate each behavior in the set of behaviors for that model according to the reasoning strategies described below. This will result in a set of inferred behaviors (note that the term generated is used in this dissertation as a synonym for inferred in the context of Simulation) with non-start-state conditions filled in with simulation-generated state conditions. Then, to do behavioral evaluation, Simulation (step 2) compares the simulation results (the generated behaviors) against the input SBF* model and determines any inconsistencies between the two as results. Simulation also (step 3) compares the simulation results against the functions in the input SBF* model to determine the extent to which those behaviors produce output expected by the functions. Issues here are also added to the results.

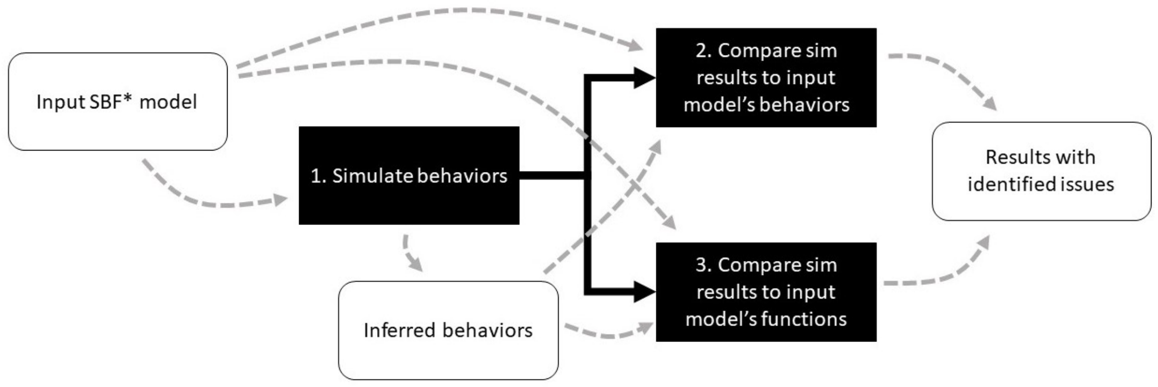

Input SBF* model
1. Simulate behaviors
Inferred behaviors
2. Compare sim results to input model's behaviors
3. Compare sim results to input model's functions
Results with identified issues

Figure 4.6: Overview of DESC's Simulation technique

The sections below provide detailed explanations of how Simulation actually simulates the behaviors and compares the outcomes of simulation to produce the final output. Algorithm descriptions are kept at a logical level because the exact technical specifics (such as what function calls are made and what data structures are used) are unnecessary to understand the technique and are implementation details. Also, the ending stage where Simulation's output is converted from an internal representation to human-readable form is also skipped because this too is an implementation detail.

4.4.1 Assumptions

Simulation makes the following assumptions regarding input given to it. The first is that the input knowledge will be represented as an SBF* model. One could expand this to be any structured representation with state transition diagrams that have the features (state conditions, explanations on transitions, equations in explanations, and function explanations) needed to simulate them according to the strategies given for Simulation. Function evaluation requires that functions exist and have provides conditions and pointers to behaviors (state diagrams) that achieve them. Zooming further out beyond these knowledge representation expectations would require changing the Simulation specifics described here. For example, perhaps one can no longer evaluation functions because there are no functions. However, the general idea of reflecting on the results of a simulation to evaluate the knowledge representation continues to make sense at an abstract level as long as simulation of the knowledge representation (or some part of it) also continues to make sense conceptually.

The second assumption made by Simulation is that the input representations are syntactically well-formed (i.e., without errors according to the rules of the representational language). If, for example an equation were not well-formed, then Simulation may either fail to resolve it, run into an error while resolving it, or may solve it incorrectly due to confusion.

Along similar lines, Simulation assumes that models do not contain any loops in their functional decompositions. A loop exists if a single function simultaneously holds subfunction and superfunction relationships (either directly or through one or more levels of indirection) to another function. Such situations cause reasoning through functions explanations in Simulation to enter an infinite looping state, meaning that the simulation will never end (or, practically, will eventually hit an error state).

Finally, the Implicit Value Forwarding feature of Simulation exists because of an assumption made about the state transition diagrams (behaviors). That is, when a previously defined attribute value in a state condition is not articulated in a future state condition, one assumes that the value continues unchanged. If this assumption is violated, one must consider what to do about Implicit Value Forwarding, perhaps ablating it from Simulation or modifying it to meet the new interpretation for missing previous attribute values in states.

4.4.2 Simulating Behaviors

To simulate a behavior, Simulation infers the attributes and values for the subsequent states of a behavior given the conditions (attribute values) in the `Start` state of that behavior. Figure 4.7 depicts a more detailed process account of behavioral simulation (step 1 above) in pseudocode. This is the top-level description with more details to be provided later in this section.

```
Input: a set of Input behaviors IB
Output: a set of generated behaviors GB
Process:
For each $IB_i$ in IB

  1. Create a copy of $IB_i$ without state conditions except for those in the Start states,
     call this $GB_i$. $GB_i$ is a generated behavior in GB

  2. Put the Start state in $GB_i$ in a stack

  3. While the stack is not empty

     (a) Pop the top of the stack. Call this the Before state
     (b) For each outgoing transition T from the Before state (in $GB_i$)
           i. Let the terminating state of T be the After state
          ii. Process the (Before state, T, After state) triple (see Figure 4.10)
         iii. Place the After state on the stack
```

Figure 4.7: Pseudocode for behavioral simulation in the Simulation technique

To simulate a behavior, Simulation first makes a copy of the input behavior and wipes out all state conditions except those in the start state. To help visualize this, the first row of Figure 4.8 depicts the `TipsResistPulloutBehavior` behavior of the Medical Patch model with its transition explanations and state names hidden. The leftmost state is the start state and the behavior proceeds from left to right. The second row depicts the behavior again, but all of its states except the Start state have their conditions removed.

Simulation then iteratively traverses outwards, simulating all outgoing transitions that it

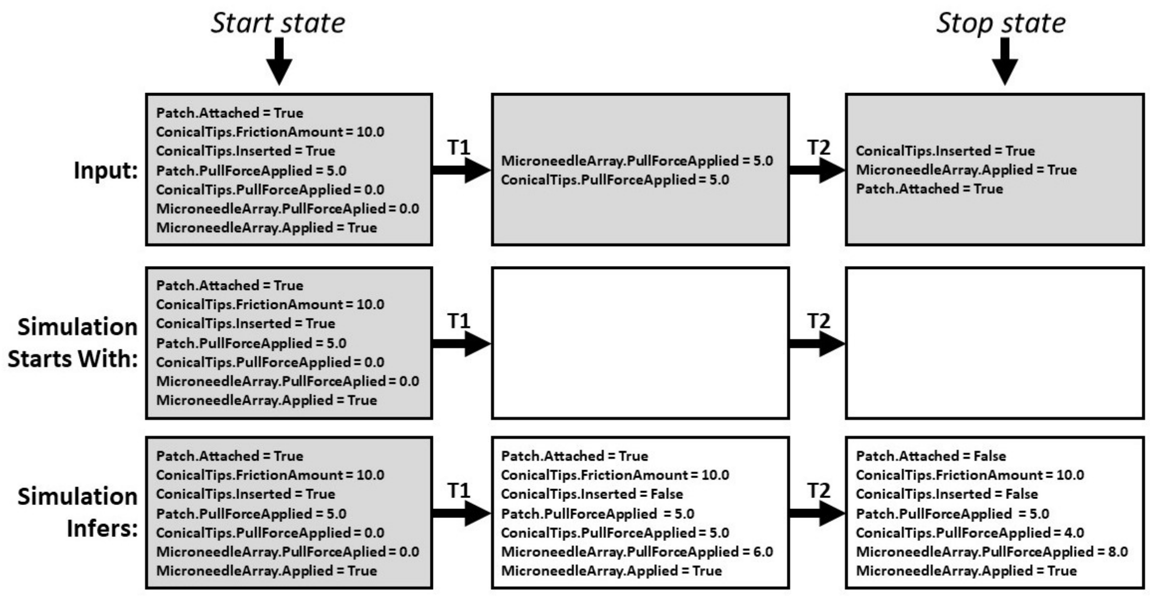
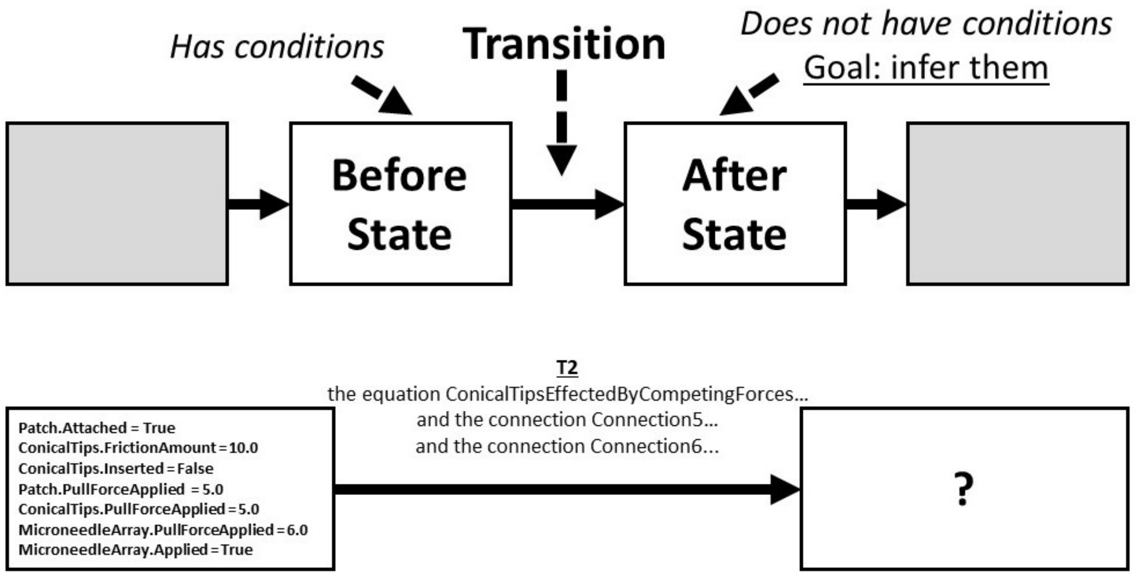

Start state
Stop state
Input:
Patch.Attached = True
ConicalTips.FrictionAmount = 10.0
ConicalTips.Inserted = True
Patch.PullForceApplied = 5.0
ConicalTips.PullForceApplied = 0.0
MicroneedleArray.PullForceAplied = 0.0
MicroneedleArray.Applied = True
T1
MicroneedleArray.PullForceApplied = 5.0
ConicalTips.PullForceApplied = 5.0
T2
ConicalTips.Inserted = True
MicroneedleArray.Applied = True
Patch.Attached = True
Simulation Starts With:
Patch.Attached = True
ConicalTips.FrictionAmount = 10.0
ConicalTips.Inserted = True
Patch.PullForceApplied = 5.0
ConicalTips.PullForceApplied = 0.0
MicroneedleArray.PullForceAplied = 0.0
MicroneedleArray.Applied = True
T1
T2
Simulation Infers:
Patch.Attached = True
ConicalTips.FrictionAmount = 10.0
ConicalTips.Inserted = True
Patch.PullForceApplied = 5.0
ConicalTips.PullForceApplied = 0.0
MicroneedleArray.PullForceAplied = 0.0
MicroneedleArray.Applied = True
T1
Patch.Attached = True
ConicalTips.FrictionAmount = 10.0
ConicalTips.Inserted = False
Patch.PullForceApplied = 5.0
ConicalTips.PullForceApplied = 5.0
MicroneedleArray.PullForceApplied = 6.0
MicroneedleArray.Applied = True
T2
Patch.Attached = False
ConicalTips.FrictionAmount = 10.0
ConicalTips.Inserted = False
Patch.PullForceApplied = 5.0
ConicalTips.PullForceApplied = 4.0
MicroneedleArray.PullForceApplied = 8.0
MicroneedleArray.Applied = True

Figure 4.8: Simplified depiction of the `TipsResistPulloutBehavior` behavior of the Medical Patch model. The Top row is the behavior as written in the Original configuration of the model. The middle row is the behavior with its non-`Start` state conditions removed. The third row is a hypothetical version of the behavior only for illustrative purposes. It illustrates conditions added to those states as if Simulation had processed the behavior

encounters until it no longer has any transitions to process. The third (bottom) row of Figure 4.8 depicts what this might look like, with the behavior now having all of it conditions filled in. Note that the filled in conditions here are hypothetical and only for the sake of example; they are not the product of actually running Simulation on this behavior.

Recall that a transition in a behavior describes the transformation of one state (here called the `Before` state) to another state (the `After` state), and a transition is annotated with zero or more of what are called explanations in the SBF* language. Each explanation clarifies why or how the system moves from the `Before` state to the `After` state. From the perspective of the simulator, an explanation may describe the upcoming value of an attribute, or it may point to a subfunction that in turn will likely set one or more attribute values.

Figure 4.9 visualizes Simulation processing a `Before` to `After` state pair. The top part depicts this in the abstract. Given a `Before` state with conditions, Simulation determines the conditions in an `After` state. The bottom imagines this (in a simplified form) for a pair of states in the `TipsResistPulloutBehavior` behavior of the Medical Patch model. The conditions in the left-most state are the same hypothetical ones from the previous figure.



Has conditions
Transition
Does not have conditions
Goal: infer them
Before State
After State
T2
the equation ConicalTipsEffectedByCompetingForces...
and the connection Connection5...
and the connection Connection6...
Patch.Attached = True
ConicalTips.FrictionAmount = 10.0
ConicalTips.Inserted = False
Patch.PullForceApplied = 5.0
ConicalTips.PullForceApplied = 5.0
MicroneedleArray.PullForceApplied = 6.0
MicroneedleArray.Applied = True
?

Figure 4.9: (Top) Abstract visualization of Simulation's goal of inferring the conditions of a `After` state given a `Before` state with conditions. (Bottom) Simplified depiction of the same process with two states in the `TipsResistPulloutBehavior` of the Medical Patch model. The conditions in the `Before` state are hypothetical and intended only for illustrative purposes

Figure 4.10 goes into detail about how a given transition is processed. In the process of simulating a transition, Simulation first reasons by function explanations, then by implicit value forwarding, and then by equations in explanations. In each step, one or more conditions may be added to the `After` state, reflecting inferred knowledge. Each one of these three categories of reasoning is covered in subsequent sections.

Note that the order of reasoning strategies here matters. Function reasoning goes first because it does not depend on any `After` state values. Next, implicit value forwarding can inspect the equation explanations proactively to determine what attributes will get set by the equations (since the left-hand side of all equations state which attribute gets assigned by it), and the function explanations have already set their values. Thus, implicit value forwarding can confidently set all unchanging attribute values in the `After` state. Finally, the equation explanations can leverage all the reasoning that came before to resolve any `After` state-referenced attributes in their equation expressions.

```
Input: a Before state; an After state; a transition T
Output: the After state with an updated set of conditions
Process:

  1. Update the After state's conditions by reasoning through all function explanations
     in T (see Figure 4.27)
  2. Update the After state's conditions by reasoning with implicit value forwarding
     (see Figure 4.29)
  3. Update the After state's conditions by reasoning with all equations in explanations
     in T (see Figure 4.11)
```

Figure 4.10: Pseudocode for `Before` state to `After` state simulation in the Simulation technique

4.4.3 Reasoning about Equations in Explanations

This section describes how DESC infers state conditions by reasoning about equations in explanations. All explanation types that support descriptions (except for the function explanation type, whose descriptions are ignored) may be annotated with Quantitative, Qualitative, or Descriptive equations.

Figure 4.11 presents pseudocode for the overall process. Essentially, DESC iterates through the set of equations. Each iteration, it solves the first equation that it encounters and deems solvable, where a solvable equation is one where the values of all references to other attributes are known. Once an equation is solved, it is removed from the set of possible equations, and the process iterates again. The process ends either when all equations are solved (a success state) or when no solvable equation can be found (an error state). When a given equation is to be solved, DESC executes a different procedure depending on if the equation is of Quantitative, Qualitative, or Descriptive type, determined by the type of the attribute being assigned to.

```
Input: a Before state; an After state; a set of explanations with equations
Output: the After state with an updated set of conditions
Process:
    1. Put all explanations in a list L
    2. While L is not empty
        (a) Let X be any one explanation in L with a solvable equation Q, meaning the
            values of all right-hand-side referenced attributes are known. If there exists
            no such X, fail and exit.
        (b) If there exists no such X, fail and exit
        (c) Solve the right-hand side of Q (see Figures 4.13 through 4.23) and create a
            condition that pairs the left-hand side's attribute with the resulting value
        (d) Remove X from L
```

Figure 4.11: Pseudocode for reasoning with equations in the Simulation technique

4.4.3.1 Reasoning about Descriptive Equations in Explanations

A component attribute can be one of three types: Quantitative, Qualitative, and Descriptive. When the attribute on the left hand side of the equation is of the Descriptive type, the equation is a descriptive equation. A Descriptive attribute is one that holds a textual value and should be reserved for those attributes with values that are neither numerical nor terms that could exist within a quantity space. For example, a `Color` attribute that holds English names for colors would be a good Descriptive attribute, for color names are non-numeric and they do not (at least not explicitly) exist within an ordered range of names.

The syntax for Descriptive equations in explanations is shown in Figure 4.12.

`<Attribute>` on the left hand side of the equation is the descriptive attribute to be set in the `After` state. `<Value>` is any chunk of text occurring in the right-hand side. This

```
eq: <Attribute> = <Value>
```

Figure 4.12: Descriptive equation syntax

text is not reasoned about by Simulation, it is merely assigned as <`Attribute`>'s value. Descriptive equation reasoning is as simple as that.

Figure 4.13 presents the pseudocode for how Simulation reasons about Descriptive equations. Given that Descriptive values are arbitrary text strings, Simulation simply assigns the value on the right-hand side of the equation to the `Attribute` on the left-hand side.

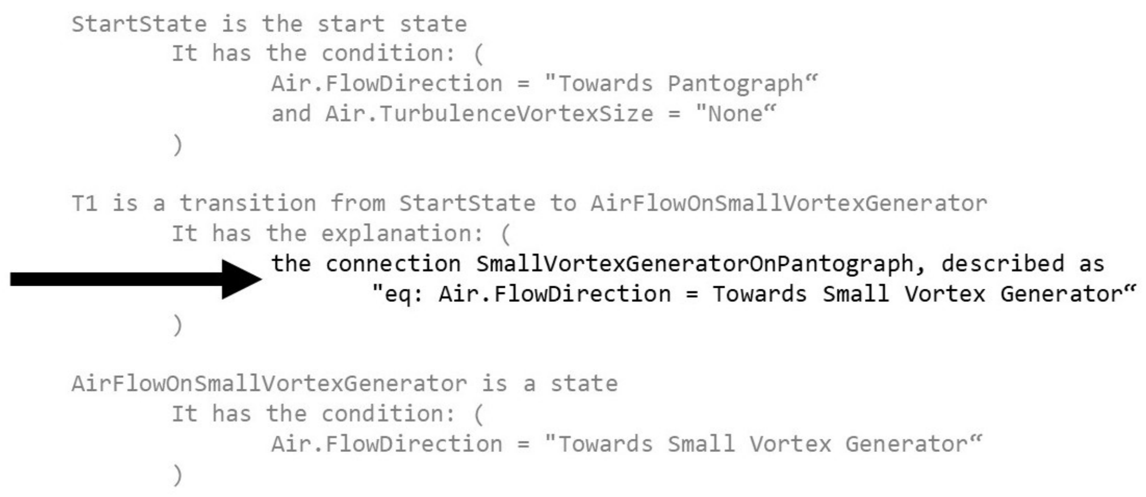
```
Input: a Before state; an After state; a descriptive equation with the syntax
    eq:  <attribute> = <any text>
Output: the After state with a new condition
Process:
  1. Set the left-hand side's <attribute> value in the After state to <any text>
```

Figure 4.13: Pseudocode for descriptive equation reasoning in the Simulation technique

Figure 4.14 depicts a snippet of the Train model where a Descriptive equation (bolded and pointed to by an arrow) is used. The Train model (see Appendix D) describes a train with small vortex generators that does not make much sound, and it is used as an example here because the Medical Patch model does not contain any Descriptive equations.

4.4.3.2 Reasoning about Quantitative Equations in Explanations

A Quantitative equation says that an `Attribute`'s value in the `After` state will be equal to the result of a mathematical expression in which variables denote component attributes that resolve to numerical values. The syntax of a Quantitative equation is shown in Figure 4.15.



```
StartState is the start state
        It has the condition: (
                Air.FlowDirection = "Towards Pantograph"
                and Air.TurbulenceVortexSize = "None"
        )

T1 is a transition from StartState to AirFlowOnSmallVortexGenerator
        It has the explanation: (
                the connection SmallVortexGeneratorOnPantograph, described as
                        "eq: Air.FlowDirection = Towards Small Vortex Generator"
        )

AirFlowOnSmallVortexGenerator is a state
        It has the condition: (
                Air.FlowDirection = "Towards Small Vortex Generator"
        )
```

Figure 4.14: Example of reasoning with descriptive equations from the Train model

```
eq: <Attribute> = <Expression>
```

Figure 4.15: Quantitative equation syntax

Here, `eq:` signifies that this is an equation to be reasoned about by Simulation. <`Attribute`> is a quantitative attribute (signifying to Simulation that the equation is quantitative) of a component to which we are assigning a value (e.g., `Box.Weight`, where `Box` is a component and `Weight` is one of its attributes). <`Expression`> refers to a mathematical expression that may contain quantitative attributes as variables. Each attribute in <`Expression`> has an additional `:Before` or `:After` tag, indicating if the value should be taken from the Before state or the After state, respectively.

Figure 4.16 presents the pseudocode for how Simulation reasons about Quantitative equations. This is a three-step process. First, Simulation replaces all Quantitative attribute references in the right-hand side of the equation with their numerical values (note that the equation will thus experience an error upon solving if any Qualitative or Descriptive attributes are referenced here because they will not be replaced). Second, Simulation mathematically solves the right-hand side of the equation. Finally, Simulation assigns that result to the attribute in the left-hand side, setting its as a condition in the `After` state.

Input: a Before state; an After state; a quantitative equation with the syntax

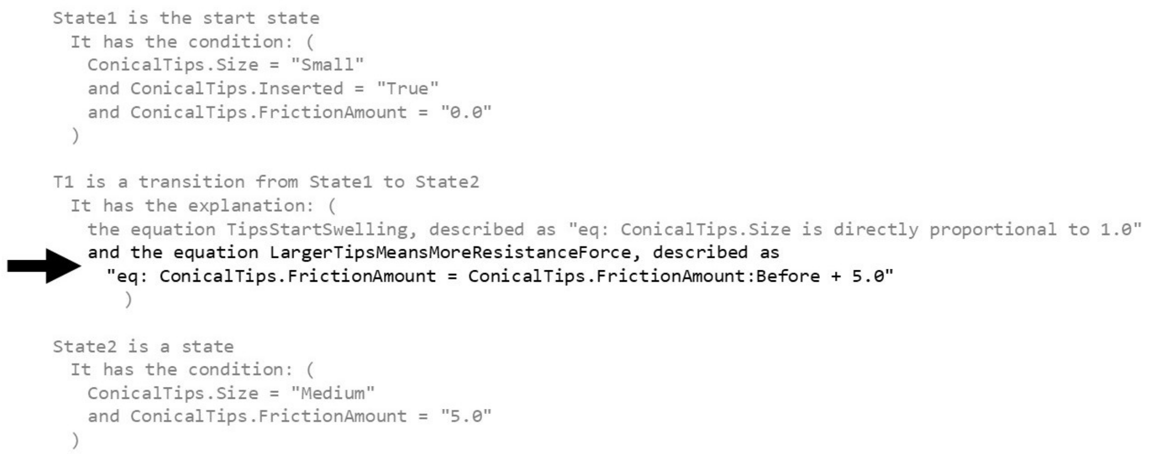
```
eq:  <attribute> = <expression>
```

Output: the After state with a new condition
Process:

1. Resolve any referenced attribute values
2. Solve <`expression`> as a mathematical equation
3. Set the left-hand side's <`attribute`> value in the After state to the result of solving <`expression`>

Figure 4.16: Pseudocode for quantitative equation reasoning in the Simulation technique

Figure 4.17 depicts a snippet of the Medical Patch model where a Quantitative equation is used. The Quantitative equation is bolded and pointed to by an arrow.



```
State1 is the start state
  It has the condition: (
    ConicalTips.Size = "Small"
    and ConicalTips.Inserted = "True"
    and ConicalTips.FrictionAmount = "0.0"
  )

T1 is a transition from State1 to State2
  It has the explanation: (
    the equation TipsStartSwelling, described as "eq: ConicalTips.Size is directly proportional to 1.0"
    and the equation LargerTipsMeansMoreResistanceForce, described as
      "eq: ConicalTips.FrictionAmount = ConicalTips.FrictionAmount:Before + 5.0"
      )

State2 is a state
  It has the condition: (
    ConicalTips.Size = "Medium"
    and ConicalTips.FrictionAmount = "5.0"
  )
```

Figure 4.17: Example of reasoning with quantitative equations from the Medical Patch model

4.4.3.3 *Reasoning about Qualitative Equations in Explanations*

In addition to reasoning with Descriptive and Quantitative equations, Simulation can also reason about Qualitative equations. Qualitative equations refer to qualitative reasoning, which is a naive reasoning strategy whereby one reasons with abstract quantities like low, medium, and high rather than precise numbers. Reasoning in this manner allows one to still consider how values change and effect each other without knowing the level of detail implied by quantitative reasoning[2]. The qualitative reasoning semantics developed for Simulation were inspired by the Garp3 technology [118] circa Summer 2011.

Quantity Spaces: Before describing how qualitative equations work, quantity spaces must be discussed. Each Qualitative-type attribute must define a quantity space in its description if that qualitative attribute is used in the function or behavior parts of the model. A quantity space is an ordered sequence of quantities, each a term that represents either a specific value (like zero) or a range of values (like medium). One orders quantities from the lowest values to the highest values (i.e., left to right in the sequence), and the quantity space captures the complete set of all possible values that the attribute may hold. For an example of a quantity space, consider the `Size` attribute of the `ConicalTips` component in the Medical Patch model. This attribute has the quantity space `Small, Medium, Large, Maximum`.

Qualitative attributes also cover when an attribute has a predefined set of values that can be expressed as a quantity space. For example, the `Inserted` attribute of the `ConicalTips` component in the Medical Patch model is Qualititative and has the quantity space `False, True`.

Simulation internally treats an attribute's quantity space as a zero-indexed array, with the first (or leftmost) quantity at index zero and going upwards from there. This provides a simple means to compare quantities and compute differences between them since smaller quantities correspond to smaller index numbers and vice versa. Thus, when Simulation converts quantities into their numerical forms, the quantity terms (or the quantity term corresponding to a referenced attribute's value) is converted to its index in the quantity

[2]That said, as the gold standard models in the Usefulness Study (chapter 6) show, one can use numbers as abstract values if needed.

space it corresponds to.

Conceptually, quantities originating from different quantity spaces should not necessarily be compared to each other because they are not actually numbers and may be on different scales. However, the term-to-number conversion done by Simulation makes this possible. One area for future work could be to explore the ramifications of doing this, such as if doing so creates opportunity for confusion.

On a related note, although one can use Qualitative attributes *or* Quantitative attributes in the right-hand side of a Qualitative equation, one cannot use both kinds of attributes simultaneously, including mixing Quantitative attributes with directly referenced Qualitative values in the `<Component>.<Attribute>.<Quantity>` construction (see below). However, one can use Qualitative attributes in addition to raw numbers (which are quantitative values), and one can also use Quantitative attributes in addition to raw numbers. Again, these decisions create opportunity for future work. Do these constraints and allowances help or hinder modeling? Is a different configuration (e.g., allowing qualitative and quantitative attributes to mix) better for model builders?

One final point about quantity spaces before discussing the Qualitative equation syntax in Simulation. Within a Qualitative equation, one can directly reference a quantity in a quantity space using the following construction: `<Component>.<Attribute>.<Quantity>`, where `<Component>` refers to a component in the structure submodel, `<Attribute>` refers to one of its attributes (which must be a qualitative attribute to be semantically correct), and `<Quantity>` refers to a quantity in that attribute's quantity space. For example, `ConicalTips.Size.Large` would refer to the `Large` quantity in of the `Size` attribute of the `ConicalTips` component in the Medical Patch model.

Qualitative Equation Forms: Qualitative equations take multiple forms in Simulation. In Form 1, the qualitative equation specifies whether an attribute's value in the `After` state is either `directly` or `inversely proportional` to a qualitative or quantitative expression. The syntax for Form 1 is shown in Figure 4.18. Here, `eq:` signifies that this is an equation to be reasoned about by Simulation, and Simulation determines that this is a qualitative equation because <`Attribute`>, which is the attribute to be changed, is Qualitative.

```
eq: <Attribute> is (directly | inversely)
    proportional to <Expression>
```

Figure 4.18: Qualitative equation syntax for Form 1

Figure 4.19 depicts pseudocode for the process by which Simulation reasons about Form 1 qualitative equations. Simulation first replaces all attributes in <`Expression`> (which must either all be Qualitative or all be Quantitative) with their numerical equivalents, and then it solves <`Expression`> as if it were a Quantitative expression. This produces a number. Simulation then inspects the result to see if it is either less than, equal to, or greater than zero. If the specified relationship is `directly proportional` and the result is greater than zero, the value of <`Attribute`> will increase in its quantity space. If equal to zero, the value will remain the same. And if less than zero, the value will decrease. If the relationship were `inversely proportional`, the increase and decrease conditions are reversed. However, the change in <`Attribute`> is limited in that an attribute's value can never increase beyond the maximum quantity in its quantity space, nor can it decrease below the minimum quantity. In such cases where an equation would cause that, the value of <`Attribute`> simply remains unchanged.

The syntax for Form 2 of Qualitative equations is shown in Figure 4.20. The syntax is a short-hand version of Form 1. Here, <`Other-Attribute`> represents a component attribute. Figure 4.21 gives the pseudocode for this form. Simulation will convert the short-hand into its Form 1 equivalent and then calculate the result in the same manner as described in Form 1's description.

Forms 1 and 2 cause the value of <`Attribute`> to increase or decrease by at most one step. Form 3 enables one to set the value to an arbitrary value in the quantity space.

Input: a Before state; an After state; a qualitative equation with the syntax

```
eq:  <attribute> is (directly | inversely) proportional
to <expression>
```

Output: the After state with a new condition
Process:

```
1. Resolve any referenced attribute values. If the attribute is Qualitative, set its
   value in the right-hand side to the quantity's numerical value
2. Solve <expression> as if it were a quantitative expression
3. If directly proportional:
   (a) If result is positive, increase the left-hand side's <attribute> value by one
       in its quantity space if possible
   (b) If result is negative, decrease the left-hand side's <attribute> value by
       one in its quantity space if possible
   (c) If result is zero, no change
4. If inversely proportional, use the same results as those under the previous
   step but reverse the increase/decrease outcomes
```

Figure 4.19: Pseudocode for qualitative equation reasoning (Form 1) in the Simulation technique

```
eq: <Attribute> is (directly | inversely)
    proportional to the change in <Other−
    Attribute>
```

Figure 4.20: Qualitative equation syntax for Form 2

Form 3's syntax is shown in Figure 4.22 and its pseudocode is in Figure 4.23. Simulation resolves <`Expression`> in the same manner described for Form 1, but it then sets the value of <`Attribute`> to whichever index in <`Attribute`>'s quantity space corresponds to the numerical result (assumed to be a plausible whole number).

Figure 4.24 depicts examples of Forms 1 and 2 of Qualitative equations, bolded and

```
Input: a Before state; an After state; a qualitative equation with the syntax

    eq: <attribute> is (directly | inversely) proportional
    to the change in <other-attribute>

Output: the After state with a new condition
Process:

  1. Convert syntax to Form 1 like so: eq: <attribute> is (directly
     | inversely) proportional to <other-attribute>:After -
     <other-attribute>:Before

  2. Process as a Form 1 Qualitative Equation.
```

Figure 4.21: Pseudocode for qualitative equation reasoning (Form 2) in the Simulation technique

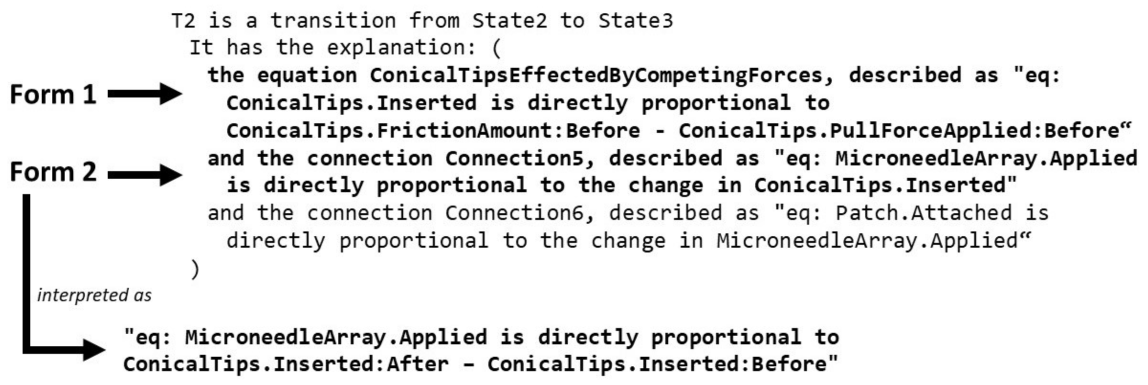
```
eq: <Attribute> is equal to <Expression>
```

Figure 4.22: Qualitative equation syntax for Form 3

```
Input: a Before state; an After state; a qualitative equation with the syntax

    eq: <attribute> is equal to <expression>

Output: the After state with a new condition
Process:

  1. Resolve any referenced attribute values. If the attribute is Qualitative, set its
     value in the right-hand side to the quantity's numerical value

  2. Solve <expression> as if it were a quantitative expression

  3. Set the left-hand side's <attribute> value to the quantity in <attribute>'s
     quantity space corresponding to the numerical result of <expression>
```

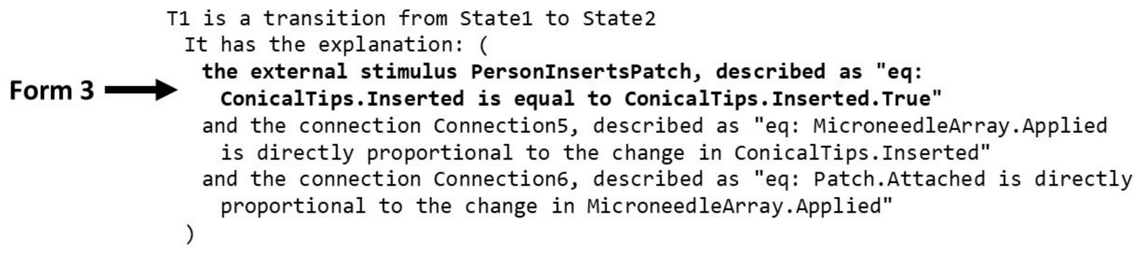
Figure 4.23: Pseudocode for qualitative equation reasoning (Form 3) in the Simulation technique

pointed to, from the Medical Patch model. The way the Form 2 equation is interpreted by Simulation is also shown. Figure 4.25 depicts an example of Form 3 of Qualitative equations, bolded and pointed to. This is also from the Medical Patch model.


Form 1
Form 2
interpreted as

```
T2 is a transition from State2 to State3
 It has the explanation: (
  the equation ConicalTipsEffectedByCompetingForces, described as "eq:
    ConicalTips.Inserted is directly proportional to
    ConicalTips.FrictionAmount:Before - ConicalTips.PullForceApplied:Before“
  and the connection Connection5, described as "eq: MicroneedleArray.Applied
    is directly proportional to the change in ConicalTips.Inserted"
  and the connection Connection6, described as "eq: Patch.Attached is
    directly proportional to the change in MicroneedleArray.Applied“
 )

"eq: MicroneedleArray.Applied is directly proportional to
ConicalTips.Inserted:After - ConicalTips.Inserted:Before"
```

Figure 4.24: Example of Form 1 and Form 2 qualitative equations from the Medical Patch model



```
T1 is a transition from State1 to State2
 It has the explanation: (
  the external stimulus PersonInsertsPatch, described as "eq:
    ConicalTips.Inserted is equal to ConicalTips.Inserted.True"
  and the connection Connection5, described as "eq: MicroneedleArray.Applied
    is directly proportional to the change in ConicalTips.Inserted"
  and the connection Connection6, described as "eq: Patch.Attached is directly
    proportional to the change in MicroneedleArray.Applied"
 )
```

Figure 4.25: Example of Form 3 qualitative equations from the Medical Patch model

4.4.4 Reasoning about Function Explanations

A function explanation on a transition in a behavior indicates that a subfunction is responsible for the change in some or all of the attribute values from the `Before` state to the `After` state. Figure 4.26, the functional decomposition of the Medical Patch model, illustrates that in SBF* modeling, behaviors mediate the functional decomposition: a behavior specifies how a function is decomposed into subfunctions.

To reason about a function explanation, Simulation effectively pauses the simulation of the current behavior then simulates the behavior linked to the function in the explanation.

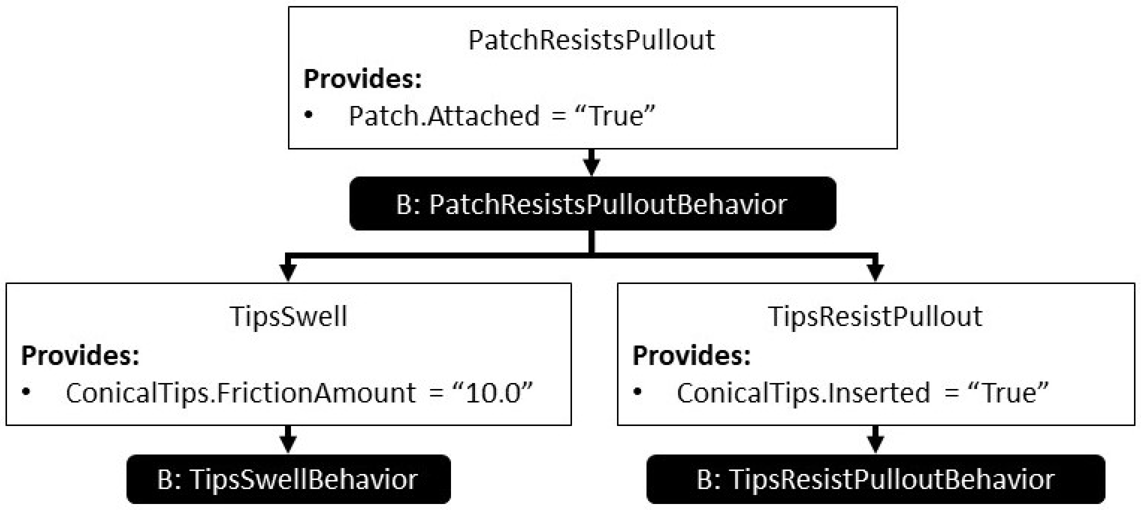

PatchResistsPullout
Provides:
• Patch.Attached = "True"
B: PatchResistsPulloutBehavior
TipsSwell
Provides:
• ConicalTips.FrictionAmount = "10.0"
B: TipsSwellBehavior
TipsResistPullout
Provides:
• ConicalTips.Inserted = "True"
B: TipsResistPulloutBehavior

Figure 4.26: Functional decomposition for the Train model with mediating behaviors

The system then sets the attribute values based on that subbehavior simulation's output and resumes simulation of the current behavior.

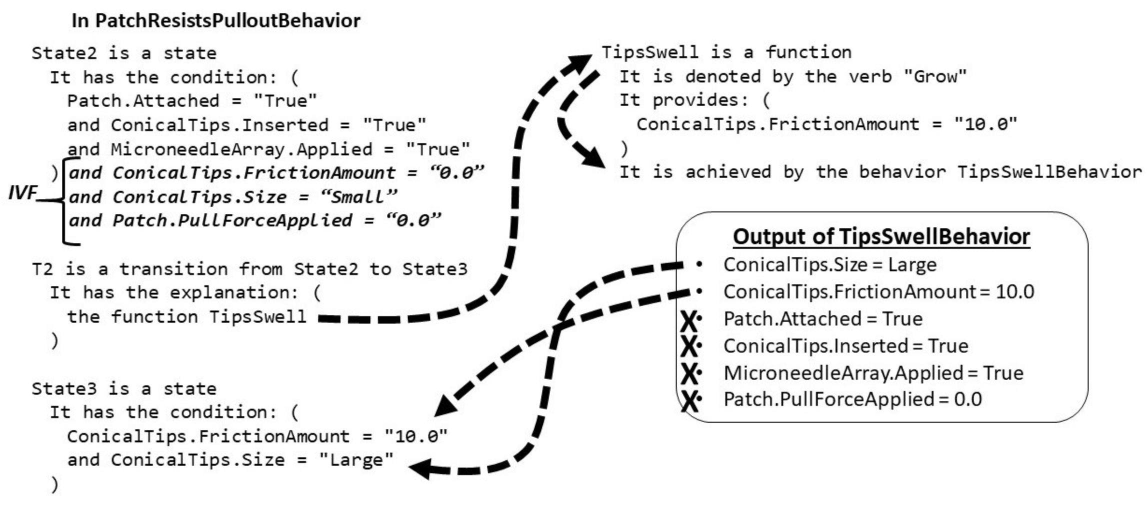
Pausing a superbehavior to simulate a subbehavior through a function explanation can be a recursive process, enabling the simulator to traverse functional decompositions of any length (albeit in practice the length is practically constrained by the implementation details). This model traversal also places a constraint on the models. If a model has a cycle in its function-subfunction decomposition, Simulation will be caught in an infinite loop and either continue forever or (more practically speaking) eventually fail due to finite computational resources. Fortunately, a cycle such as this is semantically unsound since it makes no sense for a given function to be both a superfunction and a subfunction (either directly or through some multi-step cycle) to another function, so errors arising from this situation will trigger a useful model investigation.

Figure 4.27 presents the pseudocode for function explanation reasoning in DESC. Simulation will set the `Start` state conditions values to be the superbehavior's `Before` state conditions plus any original `Start` state conditions of the subbehavior whose attributes were not covered. Simulation will then simulate the subbehavior as normal. Once done, `After` state condition will be created in the superbehavior for every attribute in the subbehavior's output whose value differs from that in the subbehavior's `Before` state or who explicitly is mentioned in the subbehavior's associated function's provides conditions. This nuanced approach was created because, if all the output conditions were applied to the superbehavior's `After` state, then a function explanation could wind up setting a lot of `After` state values coincidentally that may collide with other explanations on the superbehavior's transition being reasoned about.

```
Input: a Before state; an After state; a set of function-type explanations FX
Output: the After state with an updated set of conditions
Process:
For each function-type explanation $FX_i$ in FX
    1. Let F be $FX_i$'s function
    2. Let B be the behavior pointed to by F
    3. Simulation B per the behavior simulation process (apply process in Figure 4.7
       for one behavior), except set the Start state conditions equal to the Before state
       conditions plus any existing Start state conditions whose attributes do not appear
       in the Before state conditions
    4. Let the result of the previous step be the generated behavior's output P, defined as
       the union of all of its Stop state conditions
    5. Create a condition in the After state for each condition in P that either (i) has an
       attribute in one of F's provides conditions or (ii) has an attribute value that differs
       from the Before state's value for that attribute
```

Figure 4.27: Pseudocode for function explanation reasoning in the Simulation technique

Figure 4.28 depicts reasoning with a function explanation in the Medical Patch model. Specifically, this depicts the model transitioning from `State2` to `State3` through the transition `T2`, which contains a function explanation. The example shows how the function `TipsSwell` is pointed to and its associated behavior, `TipsSwellBehavior`, is called and then its output returned. Note how only the first two conditions in the behavior's output are returned. These represent attribute values that have changed since `State2` and/or are attributes in the function's provides conditions.



```
State2 is a state
  It has the condition: (
    Patch.Attached = "True"
    and ConicalTips.Inserted = "True"
    and MicroneedleArray.Applied = "True"
  ) and ConicalTips.FrictionAmount = "0.0"
    and ConicalTips.Size = "Small"
    and Patch.PullForceApplied = "0.0"

T2 is a transition from State2 to State3
  It has the explanation: (
    the function TipsSwell
  )

State3 is a state
  It has the condition: (
    ConicalTips.FrictionAmount = "10.0"
    and ConicalTips.Size = "Large"
  )
```

```
TipsSwell is a function
  It is denoted by the verb "Grow"
  It provides: (
    ConicalTips.FrictionAmount = "10.0"
  )
  It is achieved by the behavior TipsSwellBehavior
```

- ConicalTips.Size = Large
- ConicalTips.FrictionAmount = 10.0
- Patch.Attached = True
- ConicalTips.Inserted = True
- MicroneedleArray.Applied = True
- Patch.PullForceApplied = 0.0

Figure 4.28: Example of reasoning with a function explanation from the Medical Patch model

4.4.5 Reasoning with Implicit Value Forwarding

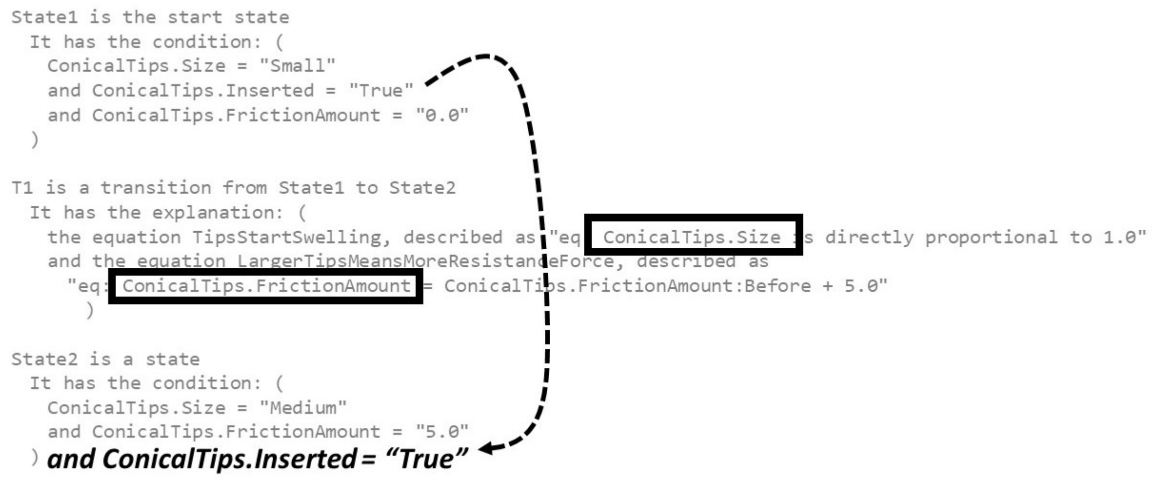
In addition to reasoning about equations in explanations and about function explanations, Simulation uses a technique here termed implicit value forwarding to address the Frame Problem in SBF* behavioral modeling. In a given `Before` and `After` state pair, Simulation may not always be able to infer `After` state values for all the attributes in the `Before` state through equation and function reasoning alone. When this is the case, it assumes that the values of those attributes remain unchanged from the `Before` state. Thus, it will set the After state values to be the same as the Before state values. This is implicit value forwarding. From a model-building perspective, this feature also simplifies modeling by not requiring that a modeler specify unchanging attribute values in every transition.

Figure 4.29 depicts the pseudocode for implicit value forwarding. Simulation will create an identical condition in the `After` state for each condition in the `Before` state whose value will not change in the transition being reasoned about.

```
Input: a Before state; an After state; the set of changing attributes (determined by
previously running function reasoning and looking ahead to determine what attributes
will get changed by equations)
Output: the After state with an updated set of conditions
Process:
For each condition C in the Before state

    1. If C's attribute is not in the set of changing attributes, create a condition in the
       After state identical to C
```

Figure 4.29: Pseudocode for implicit value forwarding in the Simulation technique

Figure 4.30 depicts reasoning with implicit value forwarding in the Medical Patch model. This figure shows `State1` transitioning to `State2` through the transition `T1`. The condition `ConicalTips.Inserted = True` is implicitly forwarded to `State2` (shown as drawn-on bold text with a dashed arrow) because Simulation could not resolve it through the two explanations on `T1`. The two explanations resolve the `ConicalTips.Size` and `ConicalTips.FrictionAmount` attributes through their associated equations. These attributes are highlighted in the equations with boxes drawn around them.



```
State1 is the start state
  It has the condition: (
    ConicalTips.Size = "Small"
    and ConicalTips.Inserted = "True"
    and ConicalTips.FrictionAmount = "0.0"
  )

T1 is a transition from State1 to State2
  It has the explanation: (
    the equation TipsStartSwelling, described as "eq: ConicalTips.Size is directly proportional to 1.0"
    and the equation LargerTipsMeansMoreResistanceForce, described as
      "eq: ConicalTips.FrictionAmount = ConicalTips.FrictionAmount:Before + 5.0"
      )

State2 is a state
  It has the condition: (
    ConicalTips.Size = "Medium"
    and ConicalTips.FrictionAmount = "5.0"
  ) and ConicalTips.Inserted = "True"
```

Figure 4.30: Example of reasoning with implicit value forwarding from the Medical Patch model

4.4.6 Evaluating the Behaviors

Evaluation of the behaviors ensures that they represent consistent descriptions of the design's processes. To this end, Simulation determines to what extent a behavior's state conditions are all justified by the transition explanations. After Simulation completes the behavior simulations, it evaluates the input behaviors. Figure 4.31 depicts the pseudocode for this process.

Simulation compares each condition in each state of each input behavior against the conditions in the matching state in the simulated or generated behavior. There will always be a known matching state because Simulation only alters the state conditions; it leaves the structure of the behavior (i.e., the states and their transitions) the same. A input condition is viewed as having an issue if, for the same component and attribute pair, there is no matching condition in the generated state or if there is a matching condition and the values differ. Note that Simulation does not view as issues if the generated state has a superset of conditions relative to the input state.

```
Input: input behaviors IB; generated behaviors GB
Output: behavior evaluation results
Process:
For each behavior $IB_i$ in IB and matching behavior $GB_i$ in GB:

    1. For each state $S_{IB}$ in $IB_i$ and matching state $S_{GB}$ in $GB_i$:

        (a) Identify as issues all conditions in $S_{IB}$ whose attribute (i) is not in $S_{GB}$'s
            conditions or (ii) is in $S_{GB}$'s conditions but with a different value
```

Figure 4.31: Pseudocode for comparing simulation results to the input model's behaviors in the Simulation technique

Figure 4.32 visualizes this for the `TipsResistPulloutBehavior` behavior of the Medical Patch model. The top part of the figure represents the behavior as given as input, and the middle part represents hypothetical (only for illustrative purposes) results from running Simulation across the behavior. The conditions of each state are compared, and differences (visualized in the dashed boxes at the bottom) are detected. Simulation has been configured only to print differences in its human-readable output that relate to conditions that appear in the input behavior, so some of these differences, while detected, will not be shown to the practitioner.

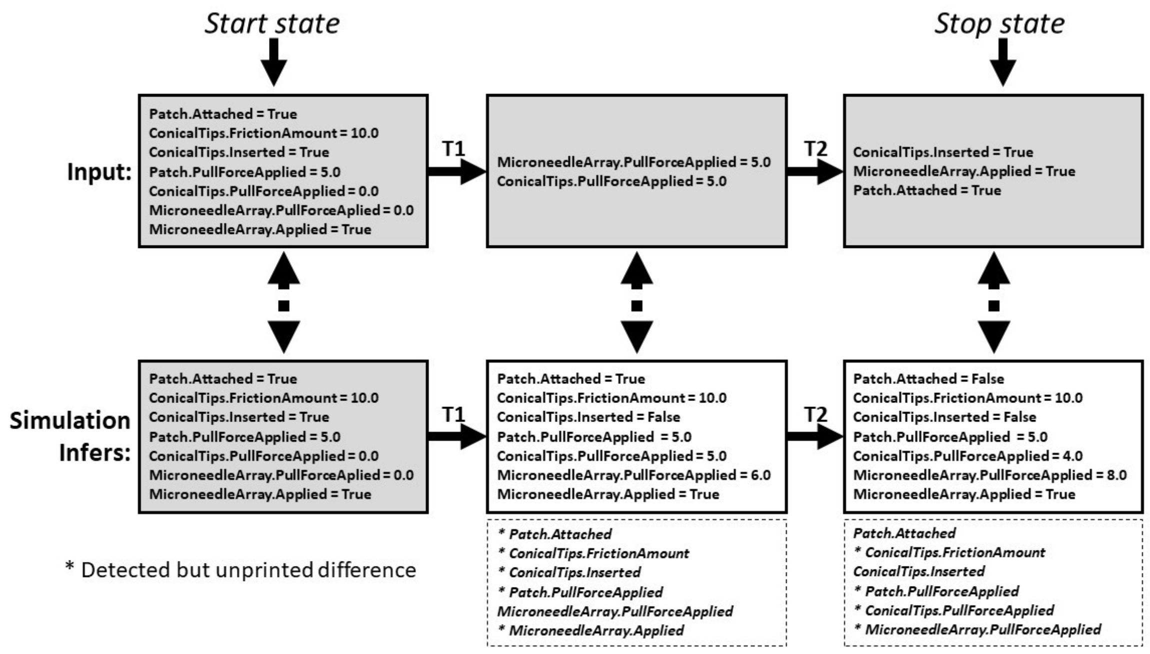

Start state
Stop state
Input:
Patch.Attached = True
ConicalTips.FrictionAmount = 10.0
ConicalTips.Inserted = True
Patch.PullForceApplied = 5.0
ConicalTips.PullForceApplied = 0.0
MicroneedleArray.PullForceAplied = 0.0
MicroneedleArray.Applied = True
T1
MicroneedleArray.PullForceApplied = 5.0
ConicalTips.PullForceApplied = 5.0
T2
ConicalTips.Inserted = True
MicroneedleArray.Applied = True
Patch.Attached = True
Simulation Infers:
Patch.Attached = True
ConicalTips.FrictionAmount = 10.0
ConicalTips.Inserted = True
Patch.PullForceApplied = 5.0
ConicalTips.PullForceApplied = 0.0
MicroneedleArray.PullForceAplied = 0.0
MicroneedleArray.Applied = True
T1
Patch.Attached = True
ConicalTips.FrictionAmount = 10.0
ConicalTips.Inserted = False
Patch.PullForceApplied = 5.0
ConicalTips.PullForceApplied = 5.0
MicroneedleArray.PullForceApplied = 6.0
MicroneedleArray.Applied = True
T2
Patch.Attached = False
ConicalTips.FrictionAmount = 10.0
ConicalTips.Inserted = False
Patch.PullForceApplied = 5.0
ConicalTips.PullForceApplied = 4.0
MicroneedleArray.PullForceApplied = 8.0
MicroneedleArray.Applied = True
* Patch.Attached
* ConicalTips.FrictionAmount
* ConicalTips.Inserted
* Patch.PullForceApplied
MicroneedleArray.PullForceApplied
* MicroneedleArray.Applied
Patch.Attached
* ConicalTips.FrictionAmount
ConicalTips.Inserted
* Patch.PullForceApplied
* ConicalTips.PullForceApplied
* MicroneedleArray.PullForceApplied
* Detected but unprinted difference

Figure 4.32: Visualizing behavior evaluation through difference detection in states. (Top) Input `TipsResistPulloutBehavior` from the Medical Patch Model. (Middle) Hypothetical (only for illustrative purposes) output of running Simulation on this behavior. (Bottom) Dashed boxes represent condition differences between the states. Some differences will not be printed in the human-readable output of Simulation to reduce clutter

Figure 4.33 depicts a reformatted snippet of the behavior results output produced by Simulation as set up for the Computational Experimentation. Specifically, these results show an example of issues detected in behavior evaluation. The model was configured as Wrong Value in a Function Provides Condition, which means a function provides condition in the inputted model was given an incorrect value. The technique (Simulation) was configured as No Function Reasoning, which means reasoning by functions was disabled. See Chapter 5 to better contextualize these terms.

- For PatchResistsPulloutBehavior:
  - For state State3:
    - ∗ You said ConicalTips.FrictionAmount = 10.0. I think that it should be 0.0
    - ∗ You said ConicalTips.Size = Large. I think that it should be Small.

Figure 4.33: Example results of behavior evaluation for the Medical Patch model from the Computational Experimentation. Model configuration was Wrong Value in a Function Provides Condition. Technique (Simulation) configuration was No Function Reasoning

4.4.7 Evaluating the Functions

Evaluation of the functions ensures that the design being modeled achieves its purposes (intended or perceived) through the behaviors. To this end, Simulation determines the extent to which the behavior responsible for achieving each function actually does achieve it according to the provides conditions associated with the function.

After Simulation has completed its behavior simulations and evaluations, it evaluates all the functions. Conceptually, the provides conditions of a function specify the attribute values that must be true at the completion of the function. Therefore, the `Stop` state of a behavior should reflect a world state that is consistent with these provides conditions.

Figure 4.34 depicts the pseudocode for this process. For a given function, Simulation compares the attribute and value pairs from the function's provides conditions with the attribute and value pairs that result from the associated simulated behavior. It looks for contradictions in these values (i.e., the output is missing a value or has a different value), thereby determining to what extent the behavior actually achieves the function. Note that attribute values that result from the behavior but are not defined by the provides conditions are ignored, for the missing attribute value pairs in the function's provides conditions are interpreted as aspects of the world state that the function does not care about.

Figure 4.35 visualizes this for the `TipsResistPullout` function and the

```
**Input:** input functions IF; generated behaviors GB
**Output:** function evaluation results
**Process:**
For each function $IF_i$ in IF:
    1. Let $GB_i$ be the behavior in GB whose matching input behavior achieves $IF_i$
    2. Let P be the output of $GB_i$, defined as the union of all of its Stop state conditions
    3. Identify as issues all conditions in $IF_i$ whose attribute (a) is not in P or (b) is in P
       but with a different value
```

Figure 4.34: Pseudocode for comparing simulation results to the input model's functions in the Simulation technique

`TipsResistPulloutBehavior` behavior that achieves it, both from the Medical Patch model. The top part of the figure represents the function as given as input, and the middle part represents hypothetical (only for illustrative purposes) results from running Simulation across the behavior. The provides conditions of the function (in this case, there is only one) are compared against the output of the inferred (or generated) behavior, which comes from its `Stop` state. Any inconsistencies between the provides conditions and the behavior's output–that is, values of attributes that differ–are reported as differences. Note that the extra attribute values in the behavior's output are not viewed as inconsistencies.

Figure 4.36 depicts a reformatted snippet of the results output produced by Simulation as set up for the Computational Experimentation. Specifically, these results show an example of an issue detected in function evaluation. The model was configured as Wrong Value in a Function Provides condition, which means a function provides condition in the inputted model was given an incorrect value. The technique (Simulation) was configured as Complete, which means all features of Simulation were enabled. See Chapter 5 to better contextualize these terms.

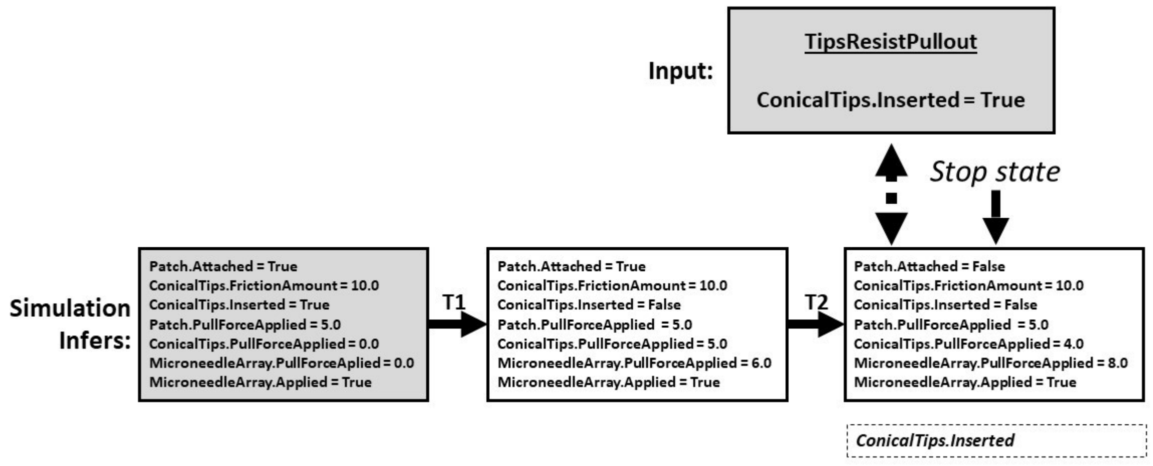

Input:
TipsResistPullout
ConicalTips.Inserted = True
Stop state
Simulation Infers:
Patch.Attached = True
ConicalTips.FrictionAmount = 10.0
ConicalTips.Inserted = True
Patch.PullForceApplied = 5.0
ConicalTips.PullForceApplied = 0.0
MicroneedleArray.PullForceAplied = 0.0
MicroneedleArray.Applied = True
T1
Patch.Attached = True
ConicalTips.FrictionAmount = 10.0
ConicalTips.Inserted = False
Patch.PullForceApplied = 5.0
ConicalTips.PullForceApplied = 5.0
MicroneedleArray.PullForceApplied = 6.0
MicroneedleArray.Applied = True
T2
Patch.Attached = False
ConicalTips.FrictionAmount = 10.0
ConicalTips.Inserted = False
Patch.PullForceApplied = 5.0
ConicalTips.PullForceApplied = 4.0
MicroneedleArray.PullForceApplied = 8.0
MicroneedleArray.Applied = True
ConicalTips.Inserted

Figure 4.35: Visualizing function evaluation through consistency evaluation of the function's provides condition against the simulated behavior's output. (Top) The function `TipsResistPullout` from the Medical Patch model and its provides condition. The behavior `TipsResistPulloutBehavior` from the same model achieves this function. (Middle) Hypothetical (only for ilustrative purposes) output of running Simulation on the `TipsResistPulloutBehavior` behavior. (Bottom) Dashed box represents that the function's provides condition was found to be inconsistent with a condition in the `Stop` state of the behavior

- For TipsSwell:
  - Your function said ConicalTips.FrictionAmount = 20.0. However, based on your associated behavior, I think that it should be 10.0.

Figure 4.36: Example results of function evaluation for the Medical Patch model from the Computational Experimentation. Model configuration was Wrong Value in a Function Provides Condition. Technique (Simulation) configuration was Complete

4.4.8 Technical Architecture

Figure 4.37 depicts the technical architecture for the Simulation technique of DESC. The boxes in this diagram represent process units, with the underlined heading categorizing them by major task. The circles represent knowledge units. The process units are represented in chronological order starting from the top although the various equation reasoners under the core algorithm could occur in any order and any given reasoner could never occur, depending on the input model. Indentation signifies that the more-indented process unit is conceptually a subprocess of the less indented one.

As with the description of and pseudocode for Simulation, the implementation description here acts as if the processing to enable multiple trajectories does not exist because this experimental feature was scoped out for this dissertation.

First, a converter translates or ingests a textual representation of an SBF* model to an internal data object representation. This converter consists of a library generated by Xtext [174] using the language definition from Appendix A. Then, the core Simulation algorithm is executed. This core algorithm leverages process units that execute different reasoning strategies. All of this produces a results internal data structure that is used by another process unit to generate Markdown [70] text for human consumption. Finally, a third-party library converts this text into HTML to be consumed by the end user. The first and last steps of this process occur outside of the core functionality of Simulation.

4.5 Comparison Technique

The other technique in DESC is Comparison. Comparison and its associated analogical mapping techniques cover the following research questions and hypotheses:

- **Research Question 3.** How might an AI agent check a design model for external consistency?

  Hypothesis 3. It may do so by analogically comparing a given model against another model of the same design, where differences represent potential issues. The Comparison technique of DESC illustrates this hypothesis.

- **Research Question 4.** Analogical comparison implies mapping. How might an AI agent map two models?

  Hypothesis 4. It may do so by decomposing the mapping task, solving the subtasks, and using solutions from earlier subtasks to constrain and inform solutions to future subtasks. Compositional Mapping illustrates this hypothesis.

Figure 4.38 diagrammatically depicts the overall process of Comparison. Given two SBF* models of the same topic (the Input model and an Alternative model), (1) align those models (a.k.a., map them) using some mapping algorithm then (2) leverage that mapping to detect difference between them. Differences between the models may signal that one of the models (or both of them!) contain misconceptions, omissions, or extraneous information at these points of difference. This process reflects an act of external consistency checking because the Alternative model acts as another opinion about the real-world concept being modeled or the vision of that concept (for the candidate design).

The Alternative model can be of any level of authority to be valuable. It need not be considered a kind of ground truth. If the Alternative model is believed to be more authoritative than the Input model, identified differences are likely issues with the Input model. For example, the Alternative model could come from a trustworthy source such as a teammate with greater expertise or from an external domain expert; or it could be automatically generated from a trustworthy source such as a scientific paper. (Motivating the plausibility of the last thought, [124]'s work automatically extracts "partial SBF models of biological systems from their natural language documents.") In these cases, the practitioner may simply take the word, so to speak, of the Alternative model and repair their Input model so that it aligns perfectly with the Alternative model. That said, more meaningful cognitive change (i.e., change in the practitioner's cognitive, internal representation of the concept that drove construction of the Input model) may occur if they investigated each difference before repairing their model to determine why the change is necessary.

If this is not the case for the Alternative model, identified differences may still be issues with the Input model, but care must be taken to vet those differences before making repairs. Perhaps the Alternative model originated from a teammate with the same or worse level of domain expertise as the practitioner who generated the Input model. In this circumstance, detected differences may reflect legitimate issues in the Input model, but they also might reflect issues in the Alternative model. The practitioner is best served by investigating the identified differences with a skeptical eye. They are potential issues, but one should only repair the Input model after confirming their authenticity and determining the correct content.

Regardless of the authority of the Alternative model, Comparison's utility is in raising areas of concern for investigation. When neither model is viewed as ground truth, problems may exist in either (or both) of them, thus differences should be deliberately addressed by the practitioner and any involved teammates to determine the best path forward. Indeed, perhaps a difference is merely a different modeling decision and does not require reconciliation.

Figure 4.39 depicts the pseudocode for the overall Comparison process. This also is a two-step process like described above. First, a map is generated between the two models and then differences are identified. How mapping works is described below. Figure 4.40 depicts the pseudocode for the difference identification step of the process. This is generalized pseudocode that does not completely apply to everything, but rather, it depicts a generic case. A model element $\mathtt{E_{IM}}$ in the Input model is identified as being a difference if (a) it is not mapped to an element in the Alternative model. Or, if (b) $\mathtt{E_{IM}}$ is mapped to some element $\mathtt{E_{AM}}$ in the other model, it is identified as a difference if $\mathtt{E_{IM}}$ and $\mathtt{E_{AM}}$ should not map (defined below and applicable only to a set of model element types) or if at least one of their associated elements are not mapped to each other. A model element $\mathtt{E_{AM}}$ in the Alternative model is identified as being a difference if it is not mapped to an element in the Input model. More reasoning is unnecessary because the Input model logic covers it.

For functions, components, attributes, attribute values, conditions, and explanations, Comparison checks if two mapped model elements *should* be mapped in detecting differences. To start, a couple assumptions: first, the mapping algorithm will only map representation elements of the same type. For example, a component should only map to another component and not to a state, transition, or something else. Second, when both the Input and Alternative models share terminology then they are referring to the same concept. That is, model elements identified by the same words in both models mean the same thing (e.g., the component `MicroneedleArray` in the Input model and the component `MicroneedleArray` in the Alternative model both refer to the same concept) and are thus identical. This greatly simplifies the task of identifying equivalence between mapped model elements.

Figure 4.41 gives the rules for determining whether two model elements should map for the types that this applies to. At an abstract level, Comparison checks the literal and semantic equivalence of two representation elements. Literal similarity asks if two representation elements (usually represented by their names) are superficially identical. That is, `MicroneedleArray` is literally identical to `MicroneedleArray` but not to `ConicalTips`.

Semantic similarity asks if the two representation elements *mean* the same thing, despite the extent to which they are literally the same. A proper implementation of semantic similarity is outside the scope of this dissertation, but a proxy form was implemented in two ways. First, when considering equivalence for aspects like component names, Comparison uses both the aforementioned literal similarity and a synthetic semantic similarity checker. This combination considers two text strings to be identical if (a) they are literally identical or (b) if they are literally identical except that one is prefixed with `Alt_` (synthetically approximating different terminology with the same meaning). For example, Comparison would consider `ConicalTips` in the Input representation to be identical to `ConicalTips` and `Alt_ConicalTips` in the Alternative representation.

Second, when considering equivalence for descriptions (such as equations) on transition explanations, Comparison will leverage prior `Component` and `Attribute` mappings to replace all `Component.Attribute` constructions in one of the descriptions with their mapped versions from the other representation. It will then run a literal equivalence check on the two descriptions. For example, assume that the component `ConicalTips` and its attribute `FrictionAmount` in the Input representation are mapped to `Alt_ConicalTips` and its attribute `Alt_FrictionAmount` in the Alternative representation[3]. The goal is to

[3]This technique is not limited to only the `Alt_` constructions.

determine if the explanation description `eq: ConicalTips.FrictionAmount = ConicalTips.FrictionAmount:Before + 5.0` in the Input representation is identical to the description `eq: Alt_ConicalTips.Alt_FrictionAmount = Alt_ConicalTips.Alt_FrictionAmount:Before + 5.0` in the Alternative representation. With only literal similarity, Comparison would say that these are not identical. Instead, Comparison converts the Input representation's description into `eq:`

```
Alt_ConicalTips.Alt_FrictionAmount =
Alt_ConicalTips.Alt_FrictionAmount:Before + 5.0
```

before checking for literal similarity between the descriptions, leading in this case to them being identical. This approach is a form of semantic similarity because `ConicalTips` and `Alt_ConicalTips` and their attributes mean the same thing (in a sense) within the context of these descriptions given the existing mapping.

Figures 4.42, 4.43, and 4.44 depict example output (reformatted for this document) of Comparison. Figure 4.42 depicts a snippet of the output for the behaviors aspect of the models, using the Behavior/Function Reorganization configuration of the Medical Patch model as the Alternative model for comparison and using the Compositional Mapping configuration of Comparison. Figure 4.43 depicts the complete functions output for this same setup. Finally, Figure 4.44 depicts the complete structure submodel results for the Combined Components configuration of the Friction Drag model with the Compositional Mapping configuration of Comparison. The Friction Drag model describes a flat plate generating drag as it moves through a fluid.

4.5.1 Assumptions

Comparison assumes two important things. First, (for the sake of this dissertation) this technique assumes that all designs are articulated as SBF* models. Two features of SBF* models are important for Comparison. They are discrete, structured representations and thus detecting differences at the model element level makes sense. Additionally, specifically with regard to Compositional Mapping (see below), SBF* as a modeling language has a rich semantics, enabling one to form logical partitions that are needed to usefully decompose the mapping problem for Compositional Mapping.

As the second assumption, Comparison assumes that there exists an Alternative model to compare against. The origin of this Alternative model is technically irrelevant to the technique because difference detection occurs the same in any case. However, as mentioned above, the user of this technique should take into consideration the relative authority of the Alternative model when interpreting Comparison's results.

Note that this second assumption makes Comparison unlikely to be useful for a novel, candidate design, for it's possible that the candidate design is unique in the world. However, in the context of team-based design, multiple teammates might articulate their vision of the candidate design, thus enabling Comparison's use. In this circumstance, external consistency checking refers to how well the articulated design accurately represents the rest of the team's (or the people within the team's) vision of the candidate design.

4.5.2 Compositional Mapping

Step 1 of the Comparison process involves forming a mapping (or alignment) between two models, and to do so, Comparison leverages a mapping algorithm. Two such algorithms were implemented in this dissertation. The first, discussed here, is a novel approach to analogical mapping that is called Compositional Mapping, which embodies Hypothesis 4 of this dissertation. The second technique is called Uniform Mapping, and it exists as a foil for Compositional Mapping. Uniform Mapping is described in the next section.

Hypothesis 4 (which Compositional Mapping illustrates) says that an AI agent may map two models by decomposing the mapping task, solving the subtasks, and using solutions from earlier subtasks to constrain and inform solutions to future subtasks. Implicit in this hypothesis is that this technique will produce superior results compared to an approach without these ideas. Good results are operationalized as those that identify only salient differences between the models (i.e., related to representation elements that are actually different between the two). Superior results thus will have more salient results and fewer non-salient results.

Figure 4.45 visualizes this mapping strategy. The large circles at the top represent the two models (a.k.a., in this case, designs) to be mapped, and the small circles within them are their model elements. At the bottom, the models have been decomposed into partitions (the larger circles at the bottom), which contain model elements (small circles) that may overlap partitions. Mapping occurs on a per-partition basis. The results of mapping earlier partitions constrain and guide the mapping process of future partitions, represented by the dashed lines.

Compositional Mapping leverages problem decomposition and constraint satisfaction to tackle the task of mapping two models. Thus, one must first conceptually decompose the knowledge representation into logical chunks, or partitions, at the language level, without knowledge of what is to be represented. Each partition represents a mapping subproblem to be solved by a subalgorithm. Next, one defines the order by which Compositional Mapping will address the subproblems. Order matters because the results of each subproblem along with the results from all prior subproblems get propagated as constraints to the next subproblem in the sequence, and subalgorithms promise to abide by the constraints given to them. Once partitions and ordering have been determined, Compositional Mapping can be used to solve all the subproblems. Once all subproblems have been solved, the global task of mapping the two knowledge representations is also solved.

Figure 4.46 depicts the pseudocode for Compositional Mapping. Note that the partitioning and ordering has already been determined and baked in to this algorithm. However, the subalgorithms that will actually solve each subproblem are provided as input and are thus configurable. Essentially, the algorithm steps through the different partition types in the defined order. To map a partition, Compositional Mapping uses a partition mapper and a set of constraints (which may be empty, as in the first partition addressed) to generate mapping results. These mapping results are then added to the set of constraints where they did not already exist. Then the algorithm moves on to the next partition in the order.

By decomposing the larger mapping problem into a sequence of subproblems, this dissertation hypothesizes two benefits[4]. First, the smaller size of the subproblems relative to the complete problem should be computationally simpler to solve since each problem has less relative complexity. Moreover, if the problem decomposition and the knowledge representation allow for it, this decomposition should aid scaling up to larger representations (in terms of the number of partition instances), since the complexity of at least some individual subproblems would likely remain relatively unchanged. For example, the decomposition created for SBF* models solves each behavior as its own subproblem. Input models could scale up the number of behaviors that they had, and although there would of course be more behavior subproblems to solve, each individual behavior subproblem would be about the same level of complexity as in a model with fewer behaviors. That said, an unsolved issue with scalability is the memory burden of tracking a growing set of propagated constraints and a growing results set. Note also that this potential benefit is not always applicable. For example, the functions (as a submodel) subalgorithm used in both Compositional Mapping configurations for the Computational Experimentation uses the complete functions submodel and thus would not benefit from this scaling concept.

Second, the implementor of Compositional Mapping can optimize the subalgorithms to best address each subproblem type. Perhaps certain subproblems require more computationally demanding solvers whereas others can achieve good results cheaply. Perhaps a strategy works well on certain subproblem types but poorly on others. Decomposing the mapping problem allows one to address situations like these on an ad hoc basis. Attempting to solve the mapping task all at once with a single algorithm, on the other hand, is less likely to facilitate localized optimization of this sort.

[4]This dissertation ignores the edge case where decomposing the larger problem somehow results in a single subproblem that is identical to it.

The results of subproblems (plus all preceding results in the sequence) are propagated forward as constraints to (a) maintain consistency across subproblem solutions and to (b) simplify future subproblem solving. To understand functional role (a), consider two theoretical assumptions on mapping. First is the notion of one-to-one mapping. One-to-one mapping is defined here as saying that a given element X in (say) the one representation will map to at most one element Y in another representation, and no other element Z in the first representation will map to Y. Second is symmetrical mapping, which is defined here as saying that if X in one representation maps to Y in another representation, then Y in that other representation will also map to X in the first representation. (In other words, mapping is bidirectional.) To adhere to both of these mapping assumptions, it is important that the subalgorithms construct an overall map of the two representations that violates neither of them.

Constraint propagation ensures that these two theoretical assumptions are not violated by enabling communication between the subalgorithms. Future subalgorithms receive past subalgorithm solutions and promise not to generate any solutions of their own that contradict the past solutions in terms of one-to-one and symmetrical mapping. (They also implicitly promise not to violate either assumption in their own solution set.) The communication is expressed in the form of constraints because these past solutions are constraining the space of possible solutions from the perspective of the future subalgorithms.

Adhering to one-to-one and symmetrical mapping in this way gives the impression of coherence in the global mapping when viewed from the outside. Final results will appear to be the product of a coordinated whole rather than from a disjointed set of subalgorithms and should thus, it is hypothesized, be more comprehensible and valuable to whoever or whatever intends to use the results.

The second functional role ((b) above) of constraint propagation is to simplify future subproblems. This is essentially a positive spin on the previous functional role. Past solutions may help to partially or completely solve a subproblem for a future subalgorithm, for the propagated results may map elements that are also present in the future subproblem. This simplifies the subproblem precisely because the subproblem thus begins partially or completely solved.

4.5.2.1 Partitioning the Knowledge Representation

The overall process of Compositional Mapping raises some questions that must be answered to implement this technique towards solving a task. How should one partition a knowledge representation? How should the subalgorithms work? In what order should the subproblems be solved? This and subsequent sections address these questions by providing guidance and discussing what choices were made for the implementation of Compositional Mapping in the Computational Experimentation of this dissertation.

To use a knowledge representation with Compositional Mapping, one needs to partition it. Each partition should be a logical unit in the knowledge representation at the language level, without assuming anything about the incoming model contents. This lets the method be generalizable beyond a specific topic or set of topics. A chunk can be subsumed by another chunk. The point here is to form the parts of the representations that will become the mapping subproblems to be solved. This step requires that the knowledge representation language used be rich enough to accommodate partitioning. SBF* is, but a simpler representation such as a basic semantic network would not be. With only entities and relationships as one's language definition, it is unclear how to partition the representation without advanced knowledge of the content being represented.

With regard to SBF* models, partitions were created along the major organizing units: the structure submodel, the functions (as a submodel), a function, the behaviors (as a submodel), a behavior, a behavioral state, and a behavioral transition. As one can see, some partitions subsume others (e.g., behaviors subsumes a behavior). This works because the different subalgorithms have different responsibilities for what to map. For example, the subalgorithm related to the behaviors only maps each behavior to another behavior. It does not map anything inside a behavior. Whereas the subalgorithm for a behavior maps states to states and transitions to transitions, but it does not map anything inside a state or a transition.

4.5.2.2 Developing the Subalgorithms

There should be one subalgorithm per knowledge representation partition type. This enables optimization of subalgorithms on a per-partition basis (as previously mentioned). A subalgorithm can work however one desires, but it must accept a set of mapping constraints as input (among anything else it wants to accept as input) and it must produce a mapping as output, which can be empty if nothing gets mapped. Further, this output mapping should not contradict any constraints given to the subalgorithm nor should the mapping violate the one-to-one and symmetrical mapping assumptions.

Only these few constraints are placed on the functioning of a subalgorithm to give subalgorithm developers large flexibility in how they solve a mapping subproblem. Furthermore, specifying how a subalgorithm should operate is at best unnecessary at this level of abstraction and is at worst presumptuous because it assumes foreknowledge of the knowledge representation partitions being addressed.

With regards to SBF* models, the Compositional Mapping configuration with interaction used the following subalgorithms during the Computational Experimentation. This set of subalgorithms was developed with the intent of getting strong results for the experimentation. That said, no claim is made that this represents the optimal set up for SBF* models. Further optimization would be a good topic for future research, especially regarding the trade off between efficiency and performance.

The subalgorithms fell broadly into three categories: analogical algorithms, a heuristic algorithm without interaction, and interactive algorithms with interaction. The analogical algorithms covered these partitions: the structure submodel, a behavior, and the functions (as a submodel). The structure submodel subalgorithm is responsible for mapping components, attributes, and connections. The behavior subalgorithm is responsible for mapping states and transitions within a single behavior. The functions (as a submodel) subalgorithm is responsible for mapping functions together but not their contents. For both behavior and functions, the subalgorithms used the Case Mapper implementation of the Structure-Mapping Engine to map those partitions. Case Mapper appears to support giving constraints to its mapping process, so the constraint satisfaction aspect of Compositional Mapping was realized while using an external tool. For the structure submodel, a heuristic approach is first used to create an initial mapping in the form of constraints to give to Case Mapper, which was used to create the partition mapping. The heuristic approach was conceptually the same as what was described earlier in this chapter about checking if two model elements are identical for difference identification in Compositional Mapping. Two elements only map if they are of the same type and are literally or semantically the same, with the same proof-of-concept implementation of semantic similarity as discussed earlier.

The second category of subalgorithms was heuristic without interaction. This was the subalgorithm corresponding to the behaviors (as a submodel) partition. This partition is responsible for mapping behaviors to each other, but it mapped nothing inside the behaviors. This subalgorithm is a case where the constraint propagation aspect of Compositional Mapping actually completely solved the subproblem, for in the implementation, the function partition was solved before the behaviors partition. In the function partition, the internals of functions are mapped. Since each function points to a behavior, this means that behaviors are also mapped. Thus, the subalgorithm for behaviors need not actually do anything. Its job was already done! Note that the behaviors subalgorithm implementation does actually create a mapping between behaviors as its output using the constraints given to it as input, but this was, upon reflection, unnecessary.

The third category of subalgorithms were heuristic with interaction. The associated partitions: a function, a state, and a transition. The function subalgorithm is responsible for mapping conditions (specifically, the provides conditions), components, attributes, and behaviors (those that are associated with the functions under analysis). The state subalgorithm is responsible for mapping conditions, components, and attributes. Components and attributes in these two partitions relate to those that appear in the conditions. Function and state subalgorithms share the same functionality for mapping their provides conditions and (state) conditions, respectively. The transition subalgorithm is responsible for mapping explanations along with the functions and connections that appear in explanations.

Each one of these subalgorithms works in a two stage process. First, the subalgorithm uses heuristics like those described for the structure submodel (except without any relationship to Case Mapper) to create a map for the partition. Second, if any unmapped elements remain that could be mapped without violating a constraint, the subalgorithm will prompt the user of Compositional Mapping (assumed to be a human) to finish mapping these elements to the extent they think is appropriate. Once two elements are mapped, they are removed from consideration so as to not allow violation of the one-to-one and symmetrical mapping assumptions. The interactive mode also prohibits the user from creating mappings that violate constraints.

This represents the complete set of subalgorithms implemented for the Compositional Mapping configuration with interaction for the computational experimentation. The Compositional Mapping (No Interaction) configuration (see Chapter 5) differs from this configuration only in that the function, state, and transition subalgorithms only implement the heuristic stage of the two stage process described above.

One can see from these descriptions that there are a variety of approaches to solving the subproblems to map SBF* partitions. This illustrates how one need not be confined to a single mapping strategy when using Compositional Mapping.

4.5.2.3 Ordering the Subproblems

The third and final question to be answered to fully realize Compositional Mapping is in what order to solve the subproblems (a.k.a., in what order to execute the subalgorithms).

Because constraint propagation is an important aspect of Compositional Mapping, it is important that the subproblems are solved in sequential order. That said, an interesting area for future work could be exploring ways to add simultaneous processing to improve Compositional Mapping's efficiency.

The implementor of a Compositional Mapping instance defines this order. In so doing, they are defining a process layer to Compositional Mapping–do this first, then that, and so on–including ensuring results get collated along the way and constraints are propagated forward. Selecting the order goes hand in hand with defining the subalgorithms, for one can start to take shortcuts in certain subalgorithms knowing that results from a previous subalgorithm will be available, such as described above with the behaviors (as a submodel) subalgorithm. Thus, it may be worthwhile to define the order before deciding on how the subalgorithms operate.

It is hard to provide general guidance for how to define the order of subalgorithms for Compositional Mapping. However, given that one of the functional roles of constraint propagation is to simplify future subproblem solving, it may be useful to consider starting with partitions whose elements are most connected to other parts of the representation. For example, the structure submodel is a good starting place for SBF* because components and attributes are present in both behavior state conditions and function provides conditions as well as being referenced in equation explanations. Results of the structure submodel mapping thus may constrain elements in both the functions and the behaviors. A top-down strategy was also used for SBF* ordering. For example, the behaviors are processed before processing within each behavior. This makes sense because the behaviors subalgorithm maps behaviors to each other. Once they are mapped, then one can map the contents of individual behaviors.

With regards to the SBF* models and Compositional Mapping, the ordering for the Computational Experimentation[5] was as follows: the structure submodel, then the functions

[5]This ordering is true for both Compositional Mapping configurations.

(starting with the submodel level), and then the behaviors (starting with the submodel level). The structure submodel is solved by a single subalgorithm. With regards to the functions, first the functions (but nothing within them) are aligned, then within each mapped function pair is aligned. With regards to behaviors, the behaviors (but nothing within them) are first aligned; then the states and the transitions (but nothing within them) within each mapped behavior pair; then the conditions of each mapped state pair; and finally, the explanations within each mapped transition pair. This ordering reflects the top-down strategy mentioned above, and upon reflection, it also reflects the strategy of giving precedence to subalgorithms in the mapping process that deal with elements that may appear in future subalgorithms. This ordering is also reflected in the pseudocode description of Compositional Mapping in Figure 4.46.

4.5.3 Uniform Mapping

Uniform Mapping is an alternative approach to Compositional Mapping for Comparison. Uniform Mapping says that the way to map two models is to provide complete models to a single algorithm that will then return results for the complete models. There are otherwise no constraints placed on this single, all-at-once algorithm except that it should produce a one-to-one, symmetrical mapping. Figure 4.47 visualizes this mapping strategy. Each large circle represents a model (a.k.a., in this case, a design), and the small circles represent elements within the models. The mapping occurs all-at-once across the whole models.

This dissertation's implementation of Uniform Mapping is essentially an interface or bridge to the server version of Case Mapper [119]. Specifically, it is a bridge to its Structure-Mapping Engine (SME) [44] aspect. (Note: this same bridge technology was also used for mapping partitions for the analogical-category subalgorithms in Compositional Mapping.) Thus, Uniform Mapping's implementation can be considered an instance of using SME with SBF* models.

There are thus three ways to think of Uniform Mapping's role in this dissertation. First, it is another approach to the mapping stage in Comparison, illustrating how the technique is not bound to a single mapping approach. Second, it is a conceptually ablated form of Compositional Mapping, lacking its problem decomposition feature and (because constraints are unnecessary without decomposition) its constraint satisfaction feature. Finally, Uniform Mapping is representative of a more conventional approach to mapping two representations.

How to represent SBF* models with Case Mapper's representational language posed an interesting research challenge, for the chosen representational schema is linked to the results. The schema (or language, so to speak) defined for this purpose is in Appendix B[6]. Figure 4.48 depicts an example of the conversion from SBF* to the Case Mapper representation for a snippet of the Medical Patch model. Getting an optimal Case Mapper representation for SBF* models was outside the scope of this dissertation, but it would make for an interesting piece of future work for exploring SME's relationship with SBF* models. That said, the results of the computational experimentation suggest that Case Mapper does reasonably well with the current schema.

Figure 4.49 depicts the pseudocode for Uniform Mapping. First, the Input and Alternative models are converted to SME-appropriate representations. SME is then ran on them, producing results. Finally, those results are converted back into SBF*-friendly representations for further reasoning.

4.5.4 Technical Architecture

Figure 4.50 depicts the technical architecture for DESC's Comparison technique with the mapping stage intentionally left as a placeholder. The boxes in this diagram and the others in this section represent process units, with the underlined heading categorizing them by major task. Note that "Determine Similarity Between Matches" is actually done by three modules: one for the structure, one or the behaviors, and one for the functions, and this process is actually the first part of the method that generates human-readable results. The

[6]The analogical sub-algorithms in Compositional Mapping also used this language and mostly the same implementation technology to convert SBF* models into this form.

circles represent knowledge units. The process units are represented in chronological order starting from the top, and indentation signifies that the more-indented process unit is conceptually a subprocess of the less indented one.

In this architecture, first a converter translates or ingests textual representations of two SBF* models to internal data object representations upon receiving the two models for comparison as input. This converter consists of a library generated by Xtext [174] using the language definition from Appendix A. Given in-memory SBF* representations, the core Comparison algorithm then begins, first generating a map between the two models. This Map declares what model elements are mapped together and leaves out any unmapped elements. Afterwards, Comparison determines whether the matched model elements should have matched (a.k.a., similarity checking), and finally generates a Markdown [70] human-readable textual set of results that identifies the model differences. In the last step of this architecture, the textual output is converted into HTML for human consumption by a third party library. The first and last steps of this process occur outside of Comparison's core functionality.

Figure 4.51 depicts the technical architecture for how Compositional Mapping fulfills the mapping stage of the overall Comparison process. Again, boxes are process units, with the underlined heading categorizing them by major task; and circles are knowledge units. Given in-memory SBF* models, Compositional Mapping addresses model partitions related to the Structure submodel, Functions, then Behaviors. This order is configurable and a submodel is skipped if either of the two input models is missing that submodel. Along the way, the process units build up and use a set of Constraints (if constraint propagation is enabled), and they build up an informal results map (not shown) that eventually gets converted into the results data object, called the Map here and in the generic architecture.

Figure 4.52 depicts the technical architecture for how Uniform Mapping fulfills the mapping stage of the overall Comparison process. Boxes are process units and circles are knowledge units. Given in-memory SBF* models, Uniform Mapping leverages a subalgorithm that maps two complete SBF* models. This subalgorithm works by first converting the models into their Case Mapper representations. (See Appendix B for the language developed to represent SBF* models for Case Mapper for this dissertation.) Constraints would also be converted for use in Case Mapper in this subalgorithm as well, but none ever exist for the Uniform Mapping technique, so this is ignored here. Then it leverages an interface that communicates with a Case Mapper server. The results of Case Mapper are eventually translated back into SBF* elements and used to produce the Map.

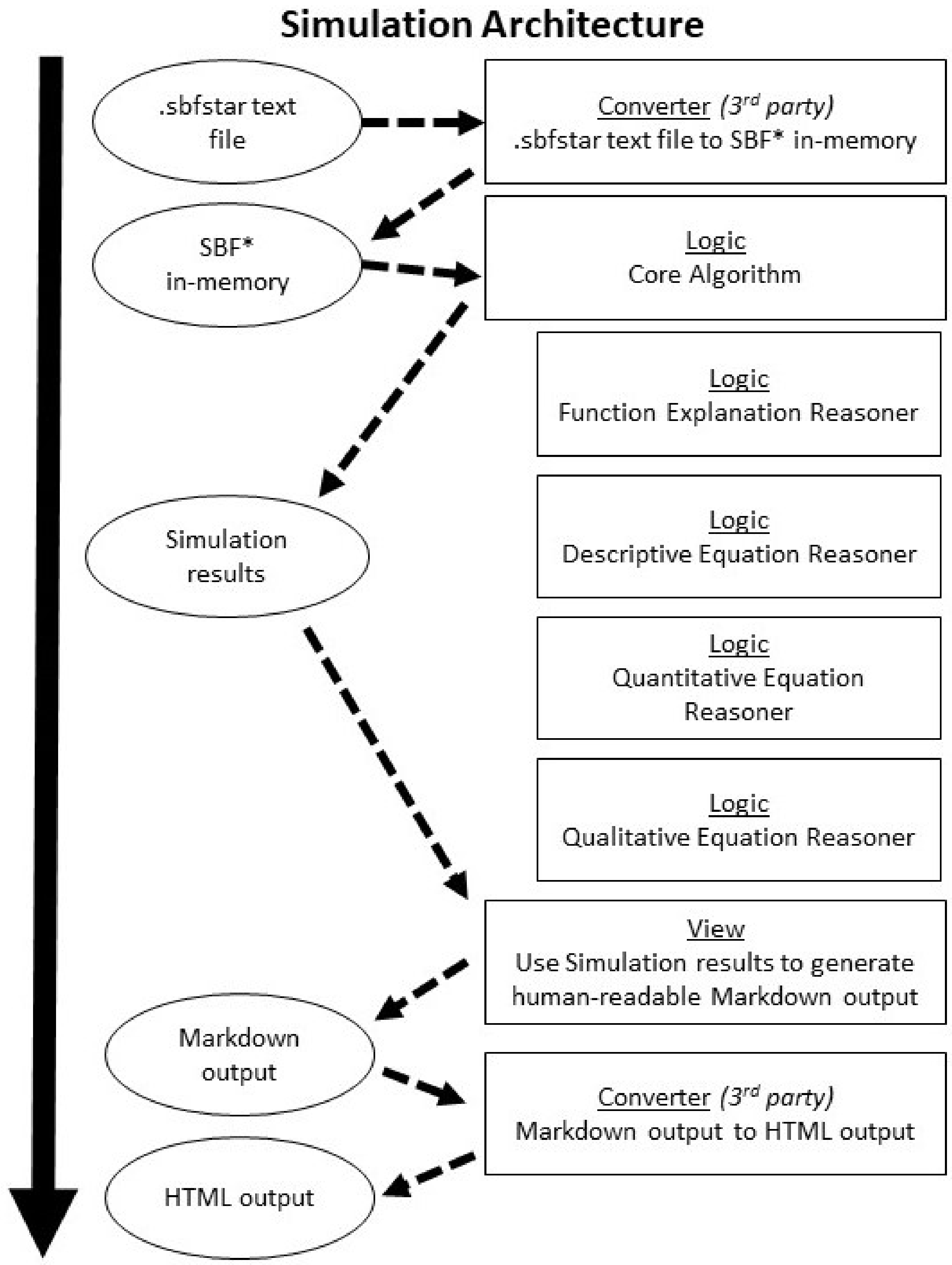

Simulation Architecture
.sbfstar text file
Converter (3rd party) .sbfstar text file to SBF* in-memory
SBF* in-memory
Logic Core Algorithm
Logic Function Explanation Reasoner
Simulation results
Logic Descriptive Equation Reasoner
Logic Quantitative Equation Reasoner
Logic Qualitative Equation Reasoner
View Use Simulation results to generate human-readable Markdown output
Markdown output
Converter (3rd party) Markdown output to HTML output
HTML output

Figure 4.37: Technical architecture for Simulation. Circles are knowledge elements. Boxes are processes

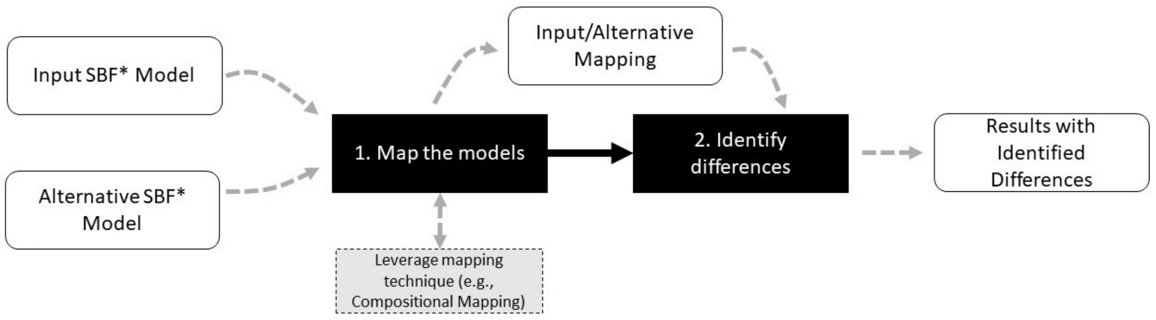

Input SBF* Model
Alternative SBF* Model
Input/Alternative Mapping
1. Map the models
2. Identify differences
Results with Identified Differences
Leverage mapping technique (e.g., Compositional Mapping)

Figure 4.38: Overview of DESC's Comparison technique

```
Input: input model IM; alternative model AM; set of partition mappers P (if using
Compositional Mapping)
Output: evaluation results for IM
Process:

  1. Generate map between IM and AM. (See Figures 4.46 and 4.49) Call this MAP

  2. Identify differences between IM and AM using MAP (See Figure 4.40)
```

Figure 4.39: Pseudocode for Comparison

```
Input: input model IM; alternative model AM; map MAP between IM and AM
Output: evaluation results for IM
Process:

  1. For each checked model element $E_{IM}$ in IM:

      (a) Identify a difference if $E_{IM}$ has no entry in MAP
      (b) OR Identify a difference if (i) $E_{IM}$ should not map with $E_{AM}$ or (ii) $E_{IM}$'s as-
          sociated elements (e.g., parent elements) are not mapped to $E_{AM}$'s associated
          elements

  2. For each model element $E_{AM}$ in AM:

      (a) Identify a difference if $E_{AM}$ has no entry in MAP
```

Figure 4.40: Pseudocode for identifying differences for Comparison (Generalized case)

Attribute values: They *should map* if one of the following is true:

- If numbers, they represent the same number.
- If not numbers, they are literally identical strings.

Conditions: They *should map* if their components, attributes, and attribute values all should map.

Explanations: They *should map* if their types are the same and one of the following is true:

- They are function type or connection type and have names that pass (i) or (ii) below
- They are not function type, do not have descriptions, and have names that pass (i) or (ii) below
- They are not function type, at least one has a description, and the descriptions are identical strings after all attribute references in one of the explanation's descriptions are replaced with mapped versions.

Components and Attributes: They *should map* if one of the following is true:

(i) Their names are literally identical strings.

(ii) Their names are semantically the same.

Figure 4.41: Rules for identifying if two model elements should map

Differences in Behaviors

- I matched your behavior PatchResistsPulloutBehavior to my behavior PatchResistsPulloutBehavior.
  - I matched your state Sub_State2 to my state State3.
    - ∗ I matched your condition ConicalTips.Size = Medium to my condition ConicalTips.Size = Large, but I do not think our values mean the same thing.
    - ∗ I matched your condition ConicalTips.FrictionAmount = 5.0 to my condition ConicalTips.FrictionAmount = 10.0, but I do not think our values mean the same thing.
  - I matched your state State3 to my state State4.
    - ∗ I did not find a match for your condition ConicalTips.FrictionAmont = 10.0.
    - ∗ I did not find a match for your condition ConicalTips.Size = Large.
    - ∗ I did not find a match for my condition Patch.PullForceApplied = 5.0.
  - I did not find a match for your state State4.
  - I matched your transition Sub_T1 to my transition T2.
    - ∗ I did not find a match for your explanation TipsStartSwelling of type Equation with description "eq: ConicalTips.Size is directly proportional to 1.0".

Figure 4.42: Differences in behaviors identified by Comparison, using the Behavior/-Function Reorganization configuration of the Medical Patch model and the Compositional Mapping configuration of the technique (Comparison)

Differences in Functions

- I matched your function PatchResistsPullout to my function PatchResistsPullout.
  - I did not find a match for your condition ConicalTips.FrictionAmount = 10.0.
- I did not find a match for my function TipsSwell.

Figure 4.43: Differences in functions identified by Comparison, using the Behavior/Function Reorganization configuration of the Medical Patch model and the Compositional Mapping configuration of the technique (Comparison)

Differences in the Structure Model

- I matched your component Flat_Plate_With_Interface to my component Flat_Plate, but I do not think they mean the same thing.
  - I did not find a match for your attribute Coefficient_for_Energy_Required_to_Transfer_Momentum
  - I did not find a match for your attribute Coefficient_for_Heat_Generated_from_Interaction
  - I did not find a match for your attribute Coefficient_for_Amount_of_Bursting_Vortices_Created
  - I did not find a match for your attribute Coefficient_for_Amount_Momentum_Transferred
- I did not find a match for my component Flat_Plate_and_Fluid_Interface.

Figure 4.44: Differences in structure submodel identified by Comparison, using the Combined Components configuration of the Friction Drag model and the Compositional Mapping configuration of the technique (Comparison)

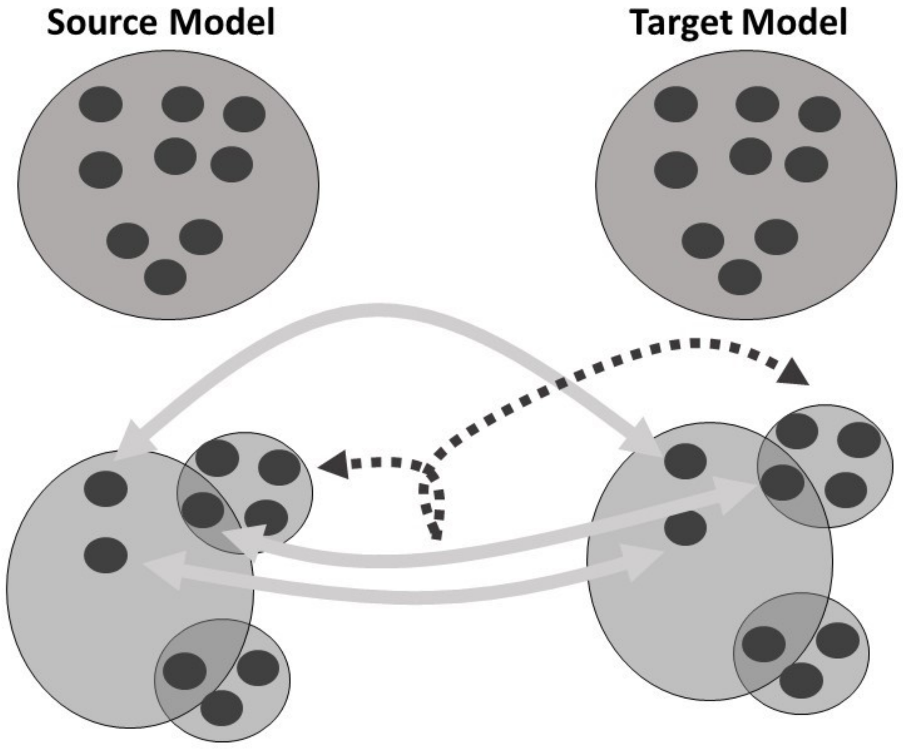


Figure 4.45: Visualization of Compositional Mapping. Given two models (top), one first decomposes the models into partitions (the larger circles at the bottom), which contain model elements that possibly overlap partitions (the smallest circles at the bottom). Mapping occurs on a per-partition basis, and the results of mapping earlier partitions constrain and guide (the dashed lines) the mapping process of future partitions

```
Input: input model IM; alternative model AM; set of partition mappers P
Output: map between IM and AM
Process:

  1. Let C be an empty set, representing constraints
  2. Map the Structure submodel of IM and AM using the $P_i$ in P for this partition
     type and given constraints C
  3. Record the elements mapped in (2) and add a constraint in C for each result if not
     already present
  4. Repeat (2) and (3) for the Functions (top-level only) of IM and AM
  5. Repeat (2) and (3) for each mapped function pair in IM and AM
  6. Repeat (2) and (3) for the Behaviors (top-level only) of IM and AM
  7. For each mapped behavior pair, $B_{IM}$ in IM and $B_{AM}$ in AM:
       (a) Repeat (2) and (3) for $B_{IM}$ and $B_{AM}$ (top-level only)
       (b) Repeat (2) and (3) for each mapped state pair in $B_{IM}$ and $B_{AM}$
       (c) Repeat (2) and (3) for each mapped transition pair in $B_{IM}$ and $B_{AM}$
```

Figure 4.46: Pseudocode for mapping two models via Compositional Mapping

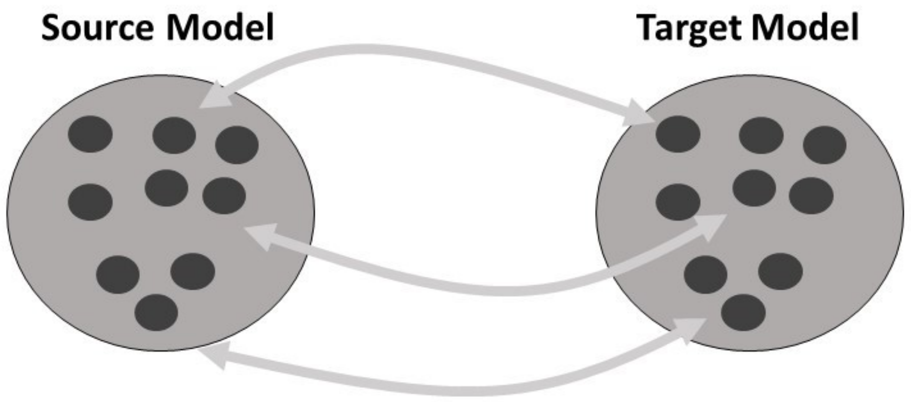
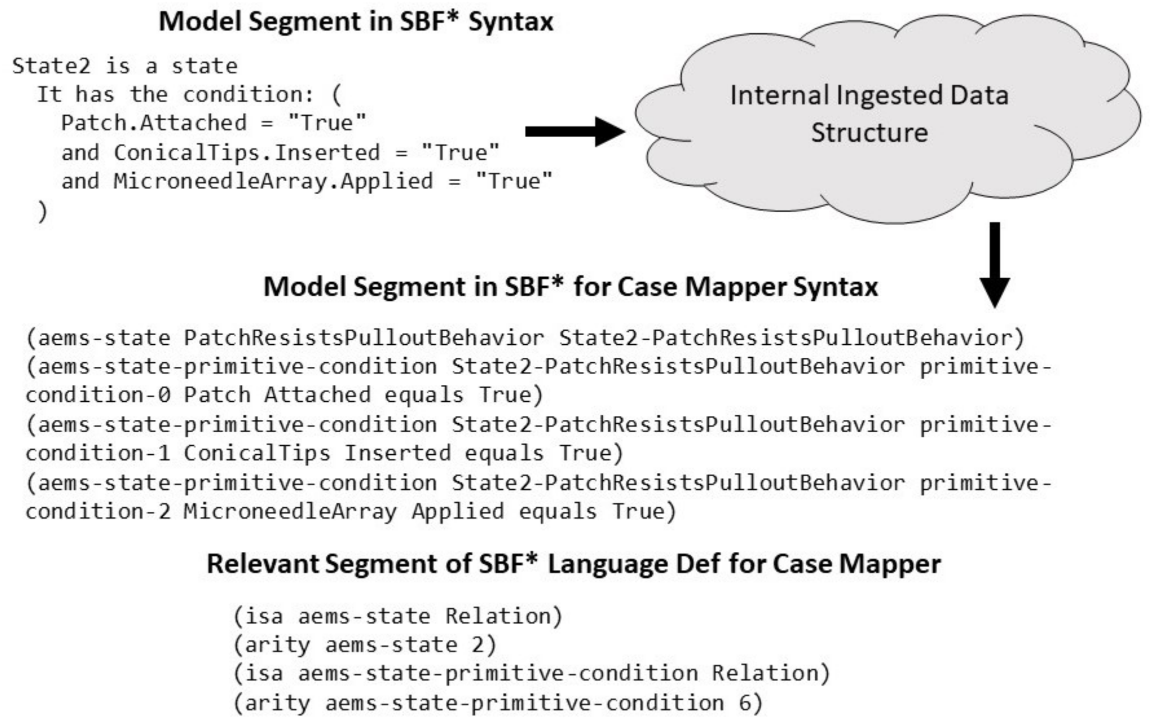


Figure 4.47: Visualization of Uniform Mapping. Each large circle represents a model, and the small circles represents elements within the models. The mapping occurs all-at-once across the entire model



Model Segment in SBF* Syntax

```
State2 is a state
  It has the condition: (
    Patch.Attached = "True"
    and ConicalTips.Inserted = "True"
    and MicroneedleArray.Applied = "True"
  )
```

Model Segment in SBF* for Case Mapper Syntax

```
(aems-state PatchResistsPulloutBehavior State2-PatchResistsPulloutBehavior)
(aems-state-primitive-condition State2-PatchResistsPulloutBehavior primitive-
condition-0 Patch Attached equals True)
(aems-state-primitive-condition State2-PatchResistsPulloutBehavior primitive-
condition-1 ConicalTips Inserted equals True)
(aems-state-primitive-condition State2-PatchResistsPulloutBehavior primitive-
condition-2 MicroneedleArray Applied equals True)
```

Relevant Segment of SBF* Language Def for Case Mapper

```
(isa aems-state Relation)
(arity aems-state 2)
(isa aems-state-primitive-condition Relation)
(arity aems-state-primitive-condition 6)
```

Figure 4.48: Example of conversion from SBF* to developed Case Mapper representation

Input: input model IM; alternative model AM
Output: map between IM and AM
Process:

1. Convert IM to SME-appropriate representation (for Case Mapper). Call this IM_{SME}
2. Convert AM to SME-appropriate representation (for Case Mapper). Call this AM_{SME}
3. Run SME (via Case Mapper) on IM_{SME} and AM_{SME}
4. Convert results to SBF*-friendly representations

Figure 4.49: Pseudocode for mapping two models via Uniform Mapping

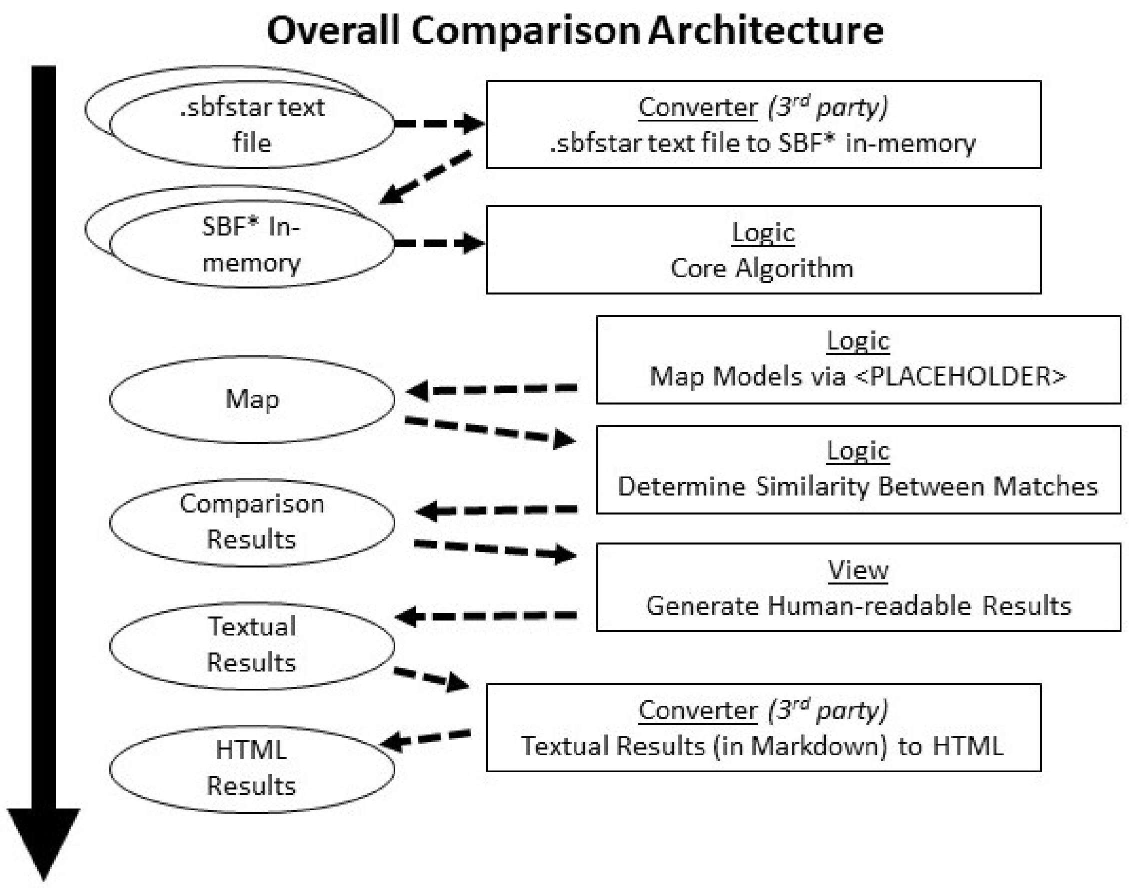

Overall Comparison Architecture
.sbfstar text file
Converter (3rd party) .sbfstar text file to SBF* in-memory
SBF* In-memory
Logic Core Algorithm
Logic Map Models via <PLACEHOLDER>
Map
Logic Determine Similarity Between Matches
Comparison Results
View Generate Human-readable Results
Textual Results
Converter (3rd party) Textual Results (in Markdown) to HTML
HTML Results

Figure 4.50: Technical architecture of Comparison (with placeholder mapping). Circles are knowledge elements. Boxes are processes

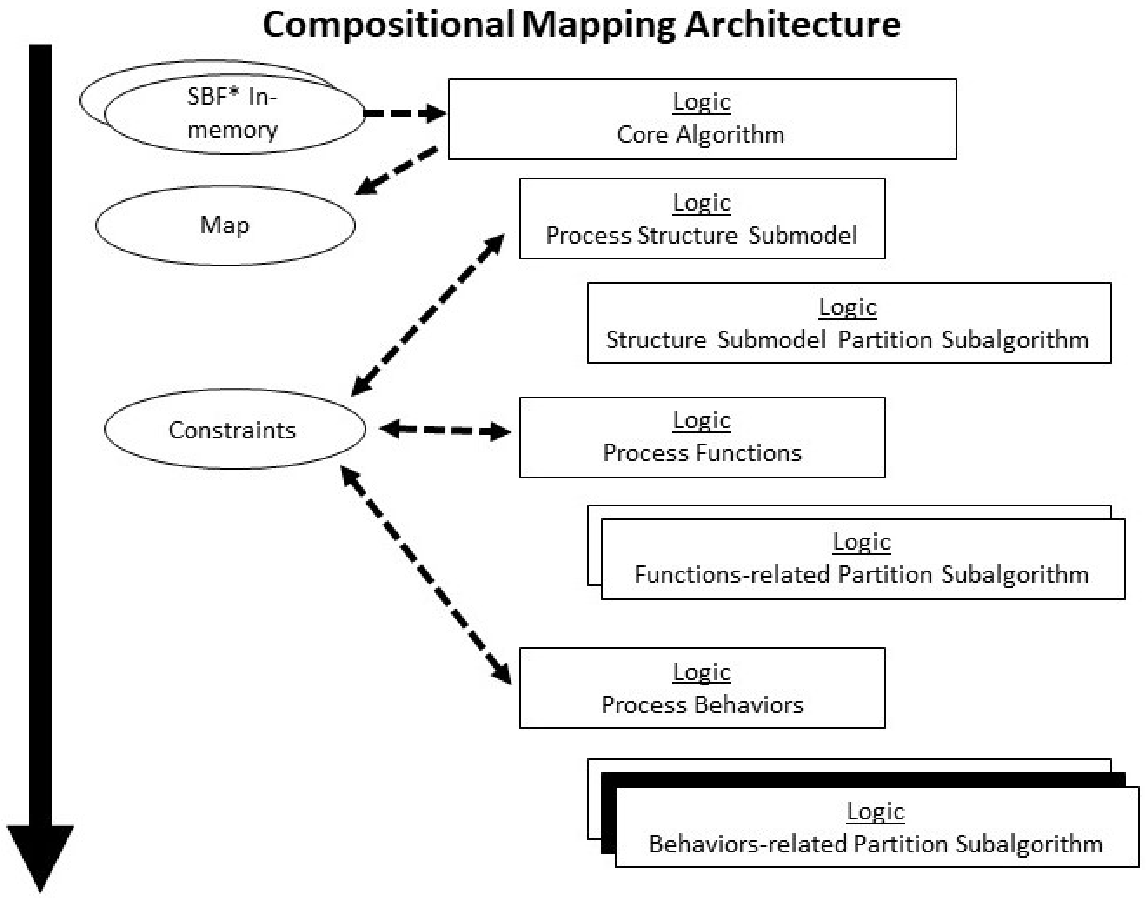

Compositional Mapping Architecture
SBF* In-memory
Map
Constraints
Logic
Core Algorithm
Logic
Process Structure Submodel
Logic
Structure Submodel Partition Subalgorithm
Logic
Process Functions
Logic
Functions-related Partition Subalgorithm
Logic
Process Behaviors
Logic
Behaviors-related Partition Subalgorithm

Figure 4.51: Technical architecture of Compositional Mapping. Circles are knowledge elements. Boxes are processes

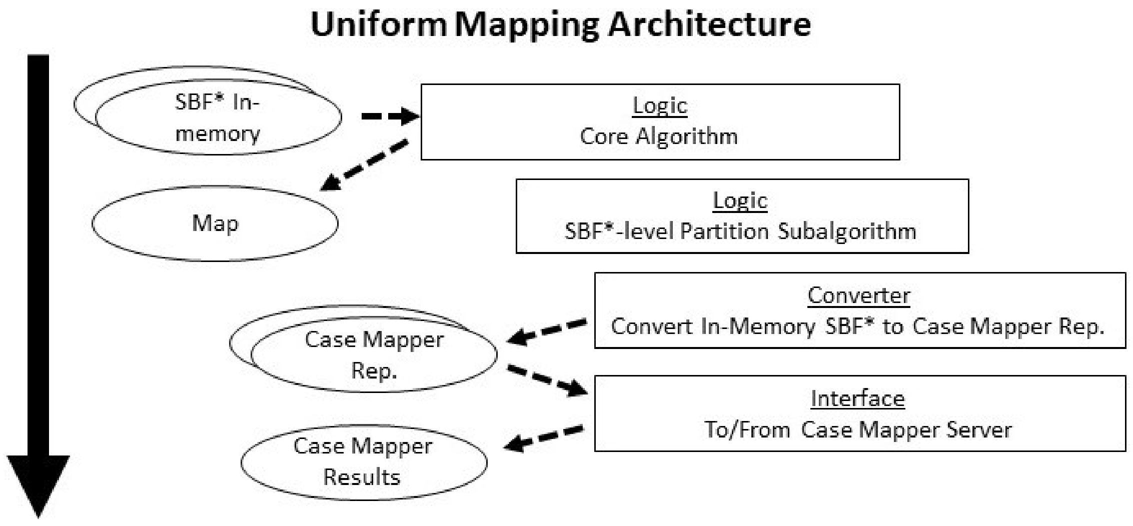

Uniform Mapping Architecture
SBF* In-memory
Logic
Core Algorithm
Map
Logic
SBF*-level Partition Subalgorithm
Converter
Convert In-Memory SBF* to Case Mapper Rep.
Case Mapper Rep.
Interface
To/From Case Mapper Server
Case Mapper Results

Figure 4.52: Technical architecture of Uniform Mapping. Circles are knowledge elements. Boxes are processes

CHAPTER V

COMPUTATIONAL EXPERIMENTATION

This chapter discusses the Computational Experimentation that was conducted with an implementation of the Design Evaluation through Simulation and Comparison (DESC) AI agent. The purpose of this experimentation is two-fold. First, it presents concrete results of applying said implementation against actual models. This provides supporting evidence that the Simulation and Comparison techniques work as expected. In doing so, the experimentation also places the techniques against concrete input and sees how they perform.

More precisely, the Computational Experimentation addresses Hypotheses 2.1, 2.2, 3, and 4 and their Research Questions. In so doing, it also addresses part of Hypothesis 1. The directly addressed hypotheses and research questions are repeated here:

- **Research Question 2.** How might an AI Agent check a design model for internal consistency?

  Hypothesis 2.1. It may do so by simulating the model and using the results to check the process and functional claims made by the model. The Simulation technique of DESC illustrates this hypothesis.

  Hypothesis 2.2. The SBF* functional modeling language provides the syntax and semantics needed for automated simulation.

- **Research Question 3.** How might an AI agent check a design model for external consistency?

  Hypothesis 3. It may do so by analogically comparing a given model against another model of the same design, where differences represent potential issues. The Comparison technique of DESC illustrates this hypothesis.

- **Research Question 4.** Analogical comparison implies mapping. How might an AI agent map two models?

  Hypothesis 4. It may do so by decomposing the mapping task, solving the subtasks, and using solutions from earlier subtasks to constrain and inform solutions to future subtasks. Compositional Mapping illustrates this hypothesis.

5.1 Design

To test the above hypotheses, computational experimentation was conducted in the form of something like an ablation study. In ablation studies, one changes the object of interest to reveal how it works underneath the black box. For this work, there are two interesting dials to turn: the techniques and the models given as input. Both were varied here (i.e., the configurations of the techniques and the models). Across a series of trials, implementations of Simulation and Comparison ran against models, and the resulting output and the time taken were both captured.

Not all changes made to the models were strictly ablations because that implies the model got worse, which was not always the case. However, the changes made to the techniques were, either directly or with Uniform Mapping as a proxy, intended as ablations in this worsening sense. Compositional Mapping with No Interaction is an exception to this, for it was created simply to better track how long Compositional Mapping takes to run. This will be discussed below.

In these trials, the techniques were ran independent of each other. While Simulation and Comparison are conceptually linked as complementary approaches within DESC to evaluating a design, neither technique's operation impacts the other. Thus, they were tested independently for simplicity. Furthermore, the reasoning in one trial does not impact other trials, so each trial should be viewed as completely independent of all others. Enabling the techniques such that they directly complement each other's reasoning and enabling learning between reasoning episodes are both interesting topics for future work.

5.1.1 Independent Variables

The independent variables are those aspects of the experimentation that were intentionally varied by the experimenter.

5.1.1.1 IV: Model Used as Input

There were six models used in this experiment: three from the Usefulness Study in Chapter 6 (Synthetic Car, Friction Drag, and Electric Toothbrush) and three more (Train, Owl Wing, and Medical Patch). Two of these models are related to biology: Friction Drag (although see below) and Owl Wing. Four are related to technology (Electric Toothbrush, Train, Medical Patch, and Synthetic Car). The Synthetic Car model is so named because it is synthetic (was not derived from source materials). One could reasonably argue that Friction Drag is actually a model of a general physics phenomenon rather than strictly a biological system, but it was derived from materials in the context of a biological system, so it is classified as such. Short descriptions and text representations of these models can be found in Appendix D under the code names used here.

To give a sense for the complexity of the models used in the Computational Experimentation, Tables 5.1, 5.2, and 5.3 describe the number of various model elements and (in the last table) the sum of all these values for each model used. Model elements range in complexity from the smallest at 38 total elements for the Owl Wing model to the largest at 149 total elements for the Electric Toothbrush model.

5.1.1.2 IV: Configuration of the Model

In addition to changing the model used as input, a given model will be set into a particular configuration to further vary the inputs given to the techniques. When not in the Original configuration, each represents an ablated or altered version of the original. The generic term configuration will be always used hereafter for simplicity.

Table 5.1: Model complexities, part 1

Model	# Behaviors	# States	# Transitions	# State Conditions	# Explanations
Synthetic Car	3	12	9	23	9
Train	5	16	11	35	19
Owl Wing	2	5	3	9	5
Medical Patch	3	11	8	32	15
Friction Drag	3	15	12	35	13
Electric Toothbrush	7	23	16	49	20

Table 5.2: Model complexities, part 2

Model	# Components	# Attributes	# Connections
Synthetic Car	5	6	0
Train	4	10	2
Owl Wing	4	4	2
Medical Patch	6	8	6
Friction Drag	5	14	0
Electric Toothbrush	6	7	0

Table 5.3: Model complexities, part 3

Model	# Functions	# Function Provides Conditions	Total Elements
Synthetic Car	3	6	76
Train	5	5	112
Owl Wing	2	2	38
Medical Patch	3	3	95
Friction Drag	3	3	103
Electric Toothbrush	7	14	149

There were two sets of configurations, one for the Simulation trials and one for the Comparison trials. These configurations, while not comprehensive for pragmatic reasons, are intended to generally address modeling situations that the two techniques address. Comparison model-configurations demonstrate different modeling decisions, whereas Simulation model-configurations demonstrate errors that the simulation can detect.

Each model underwent a subset, for brevity, of all possible model configurations. That said, all configurations were represented across the full set of trials, and every model configuration has more than one model associated with it (typically it has two). The full list of the trials is given later in this section.

Model Configurations for Comparison Trials: The Original configuration represents the original version of the model, and the other configurations reflect alternative modeling decisions.

Original: This is the model as it was intended.

Different Terminology: The names of various modeling elements are all systematically prefixed with `Alt_`, simulating a modeler using different terminology compared to the original model. The simple prefixing was chosen as an objective, straightforward way to alter the text of the model, and this standard prefix also enabled synthetic semantic similarity as described in Chapter 4. Figure 5.1 depicts both a snippet of the Original model configuration of the Electric Toothbrush model and a snippet of the same model but in the Different Terminology configuration.

Combined Components: Two components and their attributes are combined in the structure submodel, and this combination gets propagated throughout the model where appropriate to maintain internal model consistency. This configuration simulates a modeler deciding to represent the structure of the concept differently than what is given in the original model. Figure 5.2 depicts both a snippet of the Original model configuration of the Synthetic Car model and a snippet of the same model but in the Combined Components configuration.

Behavior/Function Reorganization: A subfunction and its behavior are integrated into its superfunction and superbehavior, shrinking the functional/behavioral decomposition of the model by one behavior-function pair while maintaining model consistency. This configuration simulates a modeler deciding to decompose the function and behavior aspects

```
Original_ExpertTechnologicalModel is an SBF model
  The model has a structure
    Teeth is a component
      It has a connecting point named C1
      It has an attribute Amount_of_Bacteria of type Quantitative
    On_Off_Switch is a component
      It has a connecting point named C1
      It has an attribute State of type Descriptive
    Motor is a component
      It has a connecting point named C1
      It has an attribute Continuous_Mechanical_Rotational_Energy
      of type Quantitative
```

Snippet of the Original configuration of the Electric Toothbrush model.

```
DifferentTerminology_ExpertTechnologicalModel is an SBF model
  The model has a structure
    Alt_Teeth is a component
      It has a connecting point named Alt_C1
      It has an attribute Alt_Amount_of_Bacteria of type
      Quantitative
    Alt_On_Off_Switch is a component
      It has a connecting point named Alt_C1
      It has an attribute Alt_State of type Descriptive
    Alt_Motor is a component
      It has a connecting point named Alt_C1
      It has an attribute
      Alt_Continuous_Mechanical_Rotational_Energy
      of type Quantitative
```

Snippet of the Different Terminology configuration of the Electric Toothbrush model. Meaningful changes from the Original configuration are in bold

Figure 5.1: Example for the Different Terminology model configuration

of the concept differently than the original model. Figure 5.3 depicts both a snippet of the Original model configuration of the Medical Patch model and a snippet of the same model but in the Behavior/Function Reorganization configuration, both from the function

```
Car is a component
  It has a connecting point named C1
  It has an attribute Velocity of type Quantitative
Wheels is a component
  It has a connecting point named C1
  It has an attribute RPM of type Quantitative
```

Snippet of the Original configuration of the Synthetic Car model

```
Car_With_Wheels is a component
  It has a connecting point named C1
  It has an attribute Velocity of type Quantitative
  It has an attribute RPM of type Quantitative
```

Snippet of the Combined Components configuration of the Synthetic Car model. Meaningful changes from the Original configuration are in bold

Figure 5.2: Example for the Combined Components model configuration

perspective of this configuration. Note that the `TipsSwell` function is missing in the second figure. Figure 5.4 depicts both a snippet of the Original model configuration of the Medical Patch model and a snippet of the same model but in the Behavior/Function Reorganization configuration, both from the behavior perspective of this configuration.

Figure 5.5 visualizes the Behavior/Function Reorganization configuration for the Train model in diagrammatic form. The function `EngineCausesTrainToAccelerate` and the behavior that achieves it, `EngineCausesTrainToAccelerateBehavior`, are integrated up into the superfunction `TrainGeneratesAerodynamicNoise` and superbehavior `TrainGeneratesAerodynamicNoiseBehavior`.

Model Configurations for Simulation Trials: The Original configuration represents the original version of the model, and the other configurations reflect hypothetical modeling mistakes.

Original: This is the model as it was intended. The Original model also intentionally contains no Simulation issues.

Wrong Equation: An equation in an explanation is changed. This configuration simulates a modeler cognitively reasoning about the behavior wrong, perhaps due to misunderstanding of the equation syntax or thinking about the equation incorrectly.

Figure 5.6 depicts both a snippet of the Original model configuration of the Electric Toothbrush model and the same model snippet but in the Wrong Equation configuration.

Wrong State Value: The value of an attribute in a state is changed such that it is inconsistent with the rest of the enclosing behavior. This configuration simulates a modeler cognitively reasoning about the behavior wrong, perhaps because something elsewhere was changed (like the output of a function) and this value was not updated to reflect that.

Figure 5.7 depicts a snippet of the Original model configuration of the Synthetic Car model and the same model snippet but in the Wrong State Value configuration.

Wrong Value in a Function Provides Condition: The value of an attribute in a function's provides condition is changed such that it is inconsistent with the output of the function's behavior. This configuration simulates a modeler cognitively reasoning about the function and its behavior wrong, perhaps misunderstanding the expected output of the behavior.

Figure 5.8 depicts both a snippet of the Original model configuration of the Medical Patch model and the same model snippet but in the Wrong Value in a Function Provides Condition configuration.

The sets of model configurations reflect a subset of a possible set of model differences and defects. Comparison addresses model differences. Figure 5.9 lists a speculated set of possible model differences and identifies which among these are both addressed by Comparison and tested in this Computational Experimentation. The figure organizes the taxonomy from the top down, using indentation to signify sub-categories. "Different function decomposition", "Different structure decomposition", and "Different terminology" all in the "Non-contradictory modeling decisions" category are covered by the "Combined Components", "Different Terminology", and "Behavior/Function Reorganization" model configurations, respectively.

Simulation addresses model defects. Figures 5.10 and 5.11 list a projected set of possible modeling defects and identify which among these are both addressed by Simulation and tested in this Computational Experimentation. The figure organizes the taxonomy from the top down, using indentation to signify sub-categories. The subcategory “Incorrect attribute value” in “Function satisfying” is covered by the “Wrong Value in a Function Provides Condition” model configuration. “Incorrect condition” and “Incorrect explanation”, both in “Behavior consistency”, were covered by “Wrong State Value” and “Wrong Equation” model configurations, respectively.

5.1.1.3 IV: Technique Used

A given trial in the Computational Experimentation was ran using an implementation of either the Simulation or Comparison technique of DESC.

5.1.1.4 IV: Configuration of the Evaluation Technique

The Comparison and Simulation implementations were also configured in various ways to illustrate how changes to them impact their results. Again, the techniques were tested individually, so the two sets of configurations are independent of each other.

Configurations of Simulation: The configurations here are aimed at exploring the impact of each major aspect or feature of Simulation by systematically ablating one aspect at a time.

- **Complete:** Every major feature is enabled. The configuration has function reasoning, equation reasoning, and implicit value forwarding all enabled.
- **No Function Reasoning:** This configuration has function reasoning disabled and the other two aspects enabled.
- **No Equation Reasoning:** This configuration has equation reasoning disabled and the other two aspects enabled.

No Implicit Value Forwarding: This configuration has implicit value forwarding disabled and the other two aspects enabled.

Configurations of Comparison: The configurations here are aimed at exploring the impact of changing the underlying mapping algorithm, targeting Hypothesis 4 that hypothesizes a mapping strategy illustrated by Compositional Mapping. In all cases, the subsequent step in the Comparison process that utilizes the mapping to derive the results is unchanged.

Compositional Mapping: This configuration utilizes Compositional Mapping. See Chapter 4 for a description of Compositional Mapping. The specific subalgorithm setup, which will be described later in this chapter, was chosen as an example that produces good results. Note that *optimal* results are not claimed in this dissertation.

Compositional Mapping (No Interaction): This configuration is the same as the Compositional Mapping configuration except that the subalgorithms that invoke user interaction when necessary are replaced with versions that do not invoke this interaction. This configuration was primarily used to demonstrate performance without the confounding variable of interaction time.

Uniform Mapping: The configuration utilizes Uniform Mapping. This algorithm does not use the theoretical ideas present in Compositional Mapping. Thus, Uniform Mapping acts as a proxy for a fully ablated form of Compositional Mapping and reflects a more conventional approach to analogical mapping. See Chapter 4 for a description of Uniform Mapping.

5.1.2 Dependent Variables

The dependent variables are those aspects of the experimentation that vary yet were not intentionally varied by the experimenter.

5.1.2.1 DV: Output Produced

The Computational Experimentation involves running a configured technique against a configured model (or two, in the case of Comparison trials) and capturing its output. This represents the essential dependent variable in this work. The critical issues here are (1) to what extent does the technique produce correct output (correctness) and (2) to what extent does the technique identify either the total set of expected differences or the total set of expected modeling problems (completeness).

5.1.2.2 DV: Processing Time

The amount of time each run takes was also captured. Because performance is not a central question to this dissertation, this aspect of the Computational Experimentation will receive limited coverage. Nevertheless, it does provide some insight into the feasibility of Simulation and Comparison (and thus DESC) as implemented.

5.1.3 Procedure

The Computational Experimentation has an intentionally simple and straightforward design. Comparison and Simulation techniques were tested separately because the operation and results of one does not impact the other, and trials are independent of each other. The below procedures are a simplified description of the final process and ignore situations irrelevant to the final theory experimentation such as re-runs, testing runs, or small alterations to the procedure that may have occurred due to bug fixing, implementation improvement, and theory development.

The procedure for Comparison trials was to run each trial below and then save all the results for further processing. Each trial involves running a particular technique configuration against a particular model configuration. One model is always the Original configuration of a model and the other model may be any configuration (including Original) of that same model. Note that the specific subalgorithm configuration of the Compositional Mapping configuration may require an interactive step whereby the experimenter (the dissertation author) selected an answer. When requested to intervene, the experimenter selected what they thought were reasonable answers in all cases.

The procedure for Simulation trials was to run each trial below and then save all the results for further processing. Each trial involved running a particular technique configuration against a particular model configuration.

5.1.4 Trials

The Computational Experimentation consisted of a series of trials. Below describes the trial setups.

5.1.4.1 For Comparison

Table 5.4 lists the trials for Comparison experimentation. In the trials, the source model configuration is always Original, and the same model is always compared against an alternative version of itself as the target model. The target model configurations were determined by assigning each model in a list to a random non-Original configuration and controlling the assignment such that all the model non-Original configurations were selected twice. In addition to this random assignment, two models were randomly selected to have their Original model compared against itself. All technique configurations ran on each trial. All told, 24 trials were ran across all technique and model configurations.

The trials below are ordered into logical groupings to ease understanding and make it clearer that there is equal representation across configurations. The ordering of trials does not impact the results because each trial run is independent of the others.

5.1.4.2 For Simulation

Table 5.5 lists the trials for Simulation experimentation. The model configurations were determined by assigning each model in a list to a random non-Original configuration and controlling the assignment such that all the model non-Original configurations were selected

Table 5.4: Trials for Comparison

ID	Model	Target Model Configuration
1	Electric Toothbrush	Different Terminology
2	Owl Wing	Different Terminology
3	Friction Drag	Combined Components
4	Hypothetical Car	Combined Components
5	Train	Behavior/Function Reorganization
6	Medical Patch	Behavior/Function Reorganization
7	Owl Wing	Original
8	Hypothetical Car	Original

twice. In addition to this random assignment, a trial was created for each Original model. For brevity, two of these trials (10 and 12) were randomly chosen to have all the technique configurations applied to them whereas the remaining only had the Complete technique configuration applied to them. All technique configurations were ran on all non-Original trials. All told, 36 trials were ran across all technique and model configurations.

The trials below are ordered into logical groupings to ease understanding and make it clearer that there is equal representation across configurations. The ordering of trials does not impact the results because each trial run is independent of the others.

5.1.5 Analysis Strategy

This section describes the strategies used to analyze the experiment results.

5.1.5.1 For Comparison

Comparison results analysis involved three questions. The first two were asked of each trial. First, how long did it take to run? This was computed automatically by the implementation. For trials with interactivity, the implementation also automatically calculated the duration of each interactive session. These durations were manually summed and subtracted from the total computed time to get an adjusted time that, essentially, reflects only the computational time.

Table 5.5: Trials for Simulation

ID	Model	Model Configuration
1	Electric Toothbrush	Wrong Equation
2	Owl Wing	Wrong Equation
3	Friction Drag	Wrong State Value
4	Hypothetical Car	Wrong State Value
5	Train	Wrong Value in a Function Provides Condition
6	Medical Patch	Wrong Value in a Function Provides Condition
7	Electric Toothbrush	Original*
8	Owl Wing	Original*
9	Friction Drag	Original*
10	Hypothetical Car	Original
11	Train	Original*
12	Medical Patch	Original

** Only the Complete technique configuration was run on this trial for brevity.*

Second, to what extent are the identified differences valid? At a minimum, Comparison should not detect a difference if the modeling elements (or element if unmapped) identified did not change between the Original model configuration and the configuration of the other model compared against. Thus, the differences detected by Comparison are valid if they are directly related to the model elements (i.e., involve the model elements) that were changed due to the model configuration or would be changed if they are in the Original configuration. The differences are invalid otherwise. This acts as a baseline to determine validity. Finally, to further simplify the question, comprehensiveness was not considered (since that places judgment again on what is considered a valid difference); the evaluation is simply how many invalid results that the technique produced, with the ideal state being 0 invalid results.

The third question is broader: when comparing the difference detection (second question) results between technique configurations for the same model configuration, which technique was superior? This question explores whether Compositional Mapping was superior to Uniform Mapping.

5.1.5.2 For Simulation

Simulation results analysis asked two questions of each trial. First, how long did the trial take to run? This was computed automatically by the implementation.

Second, to what extent was the set of expected model issues identical with the set of issues identified in the trial? To determine this, each non-Original model configuration used in the trials was analyzed manually (assisted by a diff tool to compare against the Original configuration) to identify the expected set of issues that the Complete configuration of Simulation should identify. Analysis of the Original model configurations was skipped because they have no Simulation-detectable issues. Next, after collecting the trial results, each result set of issues was compared against the expected set to determine if they were equal. In situations where they were not identical sets, two issues were sampled[1] and manually analyzed to verify that the results were due to the technique configurations rather than some other error. This sampling process was skipped for the Complete technique configurations.

Some of the trials failed to run to completion because the simulator hit an error. In these circumstances, neither of the aforementioned questions are applicable. Time is invalid because the simulator failed before it completed its work. Results analysis is invalid because the simulator does not produce results in this circumstance. Instead, in these cases, the model was manually inspected with the technique configuration in mind to (a) verify that the error could reasonably result from the trial set up and (b) provide reportable insight into why the error likely occurred.

5.2 Results & Analysis

This section presents results analysis for the Computational Experimentation according to the strategies presented above. Sample results are provided to ground this analysis in real

[1](i) Or one issue if only one existed. (ii) One sample each was taken from functions and behaviors if possible. (iii) Sample selection sometimes preferred those that would simplify manual analysis.

data. The results for each technique are addressed individually.

5.2.1 Simulation

The results for Simulation trials are presented here. First, the length of time it took the trials to run is reported on. Next, this document discusses the extent to which the differences detected by the trial runs were contained by the set of manually identified model differences. After that, the trials that ran into errors are reported along with thoughts on why.

5.2.1.1 Q1: Time Durations

Each trial run was automatically timed to determine how long the computer spent processing the model. Every Simulation trial run was conducted on a personal laptop (Windows 10, Intel Core i5-7200U, 8 GB RAM) in similar conditions. Across the trials, neither the mean and median amount of time (excluding runs that hit errors from these calculations) exceeded one second. This informal performance test on a personal laptop illustrates the feasibility of the Simulation technique, as this is a non-exceptional amount of running time.

These numbers do not reflect the amount of time it would take a person to analyze the output of Simulation. Future work could quantify and (if needed) refine the output presentation to improve human-reasoning speed. However, these numbers do suggest that Simulation is a reasonable approach to model evaluation since human reasoners would not be waiting any meaningful amount of time for the implementation to produce output.

5.2.1.2 Prologue to Q2: Sample Simulation Results

Figure 5.12 depicts the Function and Behavior findings for Trial 1 in the Complete technique configuration. This text, and the text for the results that follow, is slightly adjusted for formatting reasons. Header information (such as the configurations used and the run time) also present in the output has been excluded for brevity. These results are provided to give a sense for what the output looked like to contextualize the results and analysis.

These results reflect a typical case where the simulator found issues in a model. If the simulator found no issues, as in the Complete technique configuration for Trial 12, it would say what is in Figure 5.13.

If the simulator ran into an error, the error message would look something like what is depicted by Figure 5.14, which is from the No Implicit Value Forwarding technique configuration of Trial 5:

5.2.1.3 Q2: Valid/Invalid Identified Issues

As described in the analysis strategy section, the results of each trial run were manually analyzed to determine the extent to which (a) the output covered the total set of valid (or expected) model issues and (b) the output produced any invalid (or unexpected and thus wrong) model issues. Here are those results. Note that discussion of the trial runs with errors is postponed until later (Section 5.2.1.4) to simplify the discussion here.

Tables 5.6, 5.7, 5.8, and 5.9 present the results of this analysis. The tables are divided up into which model configuration type that the trials represented. Table 5.6 lists trials for the Wrong Equation model configuration; Table 5.7 lists trials for Wrong State Value; Table 5.8 for Wrong Value in a Function Provides Condition; and Table 5.9 for Original.

The results for Trial 1 in Table 5.6 show that the Complete technique configuration detects all valid issues present in the model, whereas the other technique configurations either hit an error or miss one of the issues. There were four valid issues to be found in this trial. Also, the Complete technique configuration does not detect any invalid issues, whereas the two technique configurations that run to completion do, with No Equation reasoning in particular detecting quite a few. These results make sense because the Electric Toothbrush model has both function and equation explanations so it is plausible that No Function Reasoning and No Equation Reasoning would incorrectly reason about the model and thus detect issues that are not real.

The results for Trial 2 do not follow the same pattern, but they nevertheless are consistent with the simulator's capabilities. There was one valid issue to be found in this model. The

Table 5.6: Trials 1 and 2: Valid vs invalid identified issues for the Wrong Equation model configuration

Trial	Model	Technique Configuration	All Valids Identified?	# Invalids Identified*
1	Electric Toothbrush	No Function Reasoning	No (misses 1)	11 (2 in F's, 9 in B's)
		No Equation Reasoning	No (misses 1)	34 (13 in F's, 21 in B's)
		No Implicit Value Forwarding	N/A: Hit Error	
		Complete	Yes	0
2	Owl Wing	No Function Reasoning	N/A: Hit Error	
		No Equation Reasoning	Yes	5 (2 in F's, 3 in B's)
		No Implicit Value Forwarding	Yes	0
		Complete	Yes	0

** When a non-zero value, the total is broken down into the number of issues in the Behaviors (B's) and Functions (F's)*

Complete configuration detects the valid issue and no invalid issues as expected. No Function Reasoning interestingly fails to run to completion, which is discussed in Section 5.2.1.4. No Equation Reasoning correctly detects the valid issue, but it also detects several invalid issues. The former result makes sense because the change made to the model results in a component attribute value remaining the same across states, which would happen without equation reasoning too. The latter also makes sense because this model has equations in explanations, which are being ignored. Finally, the No Implicit Value Forwarding technique configuration both correctly identifies the valid issue and finds no invalid issues. This result makes sense because, when manual analysis of the model determined that it does not require implicit value forwarding for the simulator to correctly reason about it.

The results for Trials 3 and 4 are given in Table 5.7. Both trials each only have a single valid issue to be identified. The results in both trials show a pattern similar to each other, which is that the No Implicit Value Forwarding trial runs hit errors, the Complete trial runs find the valid issue and no invalid issues, and the other technique configuration runs find the valid issue but typically through a wrong inference (more on this in a moment) and do detect invalid issues.

Table 5.7: Trials 3 and 4: Valid vs invalid identified issues for the Wrong State Value model configuration

Trial	**Model**	**Technique Configuration**	**All Valids Identified?**	**# Invalids Identified***
3	Friction Drag	No Function Reasoning	Yes	11 (2 in F's, 9 in B's)
		No Equation Reasoning	Yes (but wrong inference)	17 (3 in F's, 14 in B's)
		No Implicit Value Forwarding	N/A: Hit Error	
		Complete	Yes	0
4	Synthetic Car	No Function Reasoning	Yes (but wrong inference)	3 (1 in F's, 2 in B's)
		No Equation Reasoning	Yes (but wrong inference)	13 (5 in F's, 8 in B's)
		No Implicit Value Forwarding	N/A: Hit Error	
		Complete	Yes	0

** When a non-zero value, the total is broken down into the number of issues in the Behaviors (B's) and Functions (F's)*

The No Implicit Value Forwarding trial is addressed in Section 5.2.1.4. The Complete trial's results are as expected because it reflects the simulator with all of its features. From the perspective of the invalid results, the No Equation Reasoning and No Function Reasoning results are plausible considering that both models in these trials contain both function explanations and explanations with equations, opening the door to erroneous reasoning in these technique configurations.

When the results for the valid issue column (both in these trials and later) say that the trial run did detect the issue but had a wrong inference, this means that the output of the trial run (a) identified the issue that was expected to exist within the model, but (b) claimed that the expected value should have been something different than analysis suggests it should be. For example, in Trial 3, the value of `Flat_Plate.Relative_Velocity` in `State_2` of the behavior `Behavior_For_Flat_Plate_in_Fluid_Generates_Drag` is declared in the model to be `2.0` but it should be `1.0`. The No Equation Reasoning trial run successfully identifies this component attribute value as an issue, but it says "You said Flat_Plate.Relative_Velocity = 2.0. I think that it should be 0.0." Hence, the valid issue was identified, but the inferred value of `0.0` is wrong. That the wrong inference fit with the technique configuration was not manually checked, but in the case of this example, it is clear that the lack of equation reasoning causes the simulator to ignore an equation in a transition that would have incremented the value of `Flat_Plate.Relative_Velocity` to `1.0`, and ignoring this causes the incorrect inferred value. Incidentally, the No Function Reasoning run for trial 3 does not suffer from this issue because the single transition leading to the valid issue does not use a function explanation, and thus, the lack of function reasoning does not impact this issue.

Zooming out, these findings where the technique configuration detects the valid inference but fails to correctly infer the appropriate value illustrate that the related features of Simulation are necessary to completely support model evaluation for internal consistency checking. That is, while lacking a feature may still allow the technique to identify the flawed issue, the lack of the feature causes the simulator to suggest the *wrong* value to the user.

The results for Trials 5 and 6 are given in Table 5.8. Both trials each only have a single valid issue to be identified. Like with the previous two trials, here we see the following pattern: both No Implicit Value Forwarding trial runs hit an error, the Complete trial runs detect the valid issue and no invalid issues, and the other technique configuration runs find the valid issue but typically through a wrong inference and do detect invalid issues.

The No Implicit Value Forwarding trial is discussed in Section 5.2.1.4. The Complete trial's results are as expected because it reflects the simulator with all of its features. From the perspective of the invalid results, the No Equation Reasoning and No Function Reasoning

Table 5.8: Trials 5 and 6: Valid vs invalid identified issues for the Wrong Value in a Function Provides Condition model configuration

Trial	**Model**	**Technique Configuration**	**All Valids Identified?**	**# Invalids Identified***
5	Train	No Function Reasoning	Yes (but wrong inference)	14 (1 in F's, 13 in B's)
		No Equation Reasoning	Yes (but wrong inference)	23 (4 in F's, 19 in B's)
		No Implicit Value Forwarding	N/A: Hit Error	
		Complete	Yes	0
6	Medical Patch	No Function Reasoning	Yes	2 (2 in B's)
		No Equation Reasoning	Yes (but wrong inference)	14 (1 in F's, 13 in B's)
		No Implicit Value Forwarding	N/A: Hit Error	
		Complete	Yes	0

** When a non-zero value, the total is broken down into the number of issues in the Behaviors (B's) and Functions (F's)*

results are plausible considering that both models in these trials contain both function explanations and explanations with equations, opening the door to erroneous reasoning in these technique configurations. See above for what it means when a trial run found the valid issue but made the wrong inference about its component attribute value.

The results for Trials 7 through 12 are given in Table 5.9. Unlike the prior sets of results, all of these trials ran against models in the Original configuration, which means that they have no valid issues. Thus, this column always has N/A since the set of valid issues for every trial is empty. Also note that, for brevity, only the Complete technique configuration was run for trials 7 through 9 and also for 11.

For all trials 7 through 12, the Complete technique configuration found no invalid issues. This is expected since the Complete configuration represents Simulation with all of its features enabled.

For trials 10 and 12, a similar pattern is seen with the other technique configurations

Table 5.9: Trials 7 through 12: Valid vs invalid identified issues for the Original model configuration

Trial	Model	Technique Configuration	All Valids Identified?**	# Invalids Identified***
7*	Electric Toothbrush	Complete	N/A	0
8*	Owl Wing	Complete	N/A	0
9*	Friction Drag	Complete	N/A	0
10	Synthetic Car	No Function Reasoning	N/A	4 (1 in F's, 3 in B's)
		No Equation Reasoning	N/A	14 (5 in F's, 9 in B's)
		No Implicit Value Forwarding	N/A: Hit Error	
		Complete	N/A	0
11*	Train	Complete	N/A	0
12	Medical Patch	No Function Reasoning	N/A	2 (2 in B's)
		No Equation Reasoning	N/A	15 (2 in F's, 13 in B's)
		No Implicit Value Forwarding	N/A: Hit Error	
		Complete	N/A	0

** Only the Complete technique configuration was run on this trial for brevity.*
*** Original models have no valid issues, so the result for this column is always N/A.*
**** When a non-zero value, the total is broken down into the number of issues in the Behaviors (B's) and Functions (F's)*

as previously encountered. The No Implicit Value Forwarding trial runs hit errors, and both the No Function Reasoning and No Equation Reasoning trial runs find invalid issues. For the latter findings, these are plausible because both the Synthetic Car and Medical Patch models have both function explanations and explanations with equations, potentially allowing erroneous reasoning in those technique configurations.

5.2.1.4 *Trial Runs with Errors*

Table 5.10 lists the trial runs that failed due to running into an error. The error displayed for each of these runs was manually confirmed to be reasonable considering the absence of the feature in the given technique configuration. A few points are worthy of note. First, nearly all of the runs listed in the aforementioned table are from a No Implicit Value Forwarding technique configuration run. In fact, nearly all trials that had the No Implicit Value Forwarding technique configuration (recall trials 7-9 and 11 did not) hit an error. Trial 2 is an exception to this, discussed below.

Table 5.10: Trials with errors

Trial	Model and Model Configuration	Technique Configuration
2	Owl Wing, Wrong Equation	No Function Reasoning
1	Electric Toothbrush, Wrong Equation	No Implicit Value Forwarding
3	Friction Drag, Wrong State Value	
4	Synthetic Car, Wrong State Value	
5	Train, Wrong Value in a Function Provides Condition	
6	Medical Patch, Wrong Value in a Function Provides Condition	
10	Synthetic Car, Original	
12	Medical Patch, Original	

Why is No Implicit Value Forwarding fertile ground for an error? The absence of the implicit value forwarding feature of Simulation means that only those component attribute values implied by the equations in explanations and the function explanations will be created in the subsequent states, meaning that it is likely for one or more component attribute values to drop. Generally speaking, this creates a situation where an equation in an explanation may be unresolvable due to missing one or more needed values that would have been forwarded. This in turn means the simulator cannot complete simulation of the model, which causes the reported error.

The run from trial 2 is an interesting exception. Here, uniquely, the run has a No Function Reasoning technique configuration. Why is this the case? Here, the function explanation call creates a component attribute value that is needed for the equation in the same transition. By ignoring the function explanation, the simulator thus does not create this value, which makes the equation unresolvable and leads to an error. Also note that trial 2 does not have an error for Implicit Value Forwarding. Manually inspection of the model confirmed that the specific description of the behaviors in this model makes it so that implicit value forwarding in fact is not necessary for successfully simulating this model.

5.2.2 Comparison

The results for Comparison trials are presented here. First, the length of time it took the trials to run is reported. Next, the document discuses the trials that ran into errors. Finally, the extent to which the non-erroneous trials identified the expected model issues is analyzed.

5.2.2.1 Q1: Time Durations

Each trial run was automatically timed to determine how long the computer spent processing the model. Every Comparison trial run was conducted on a personal laptop (Windows 10, Intel Core i5-7200U, 8 GB RAM) in similar conditions. Across the trials, the largest mean and median amount of time for a trial (combining technique runs and excluding any interaction time) was about 11 seconds and about 6 seconds, respectively. Trial 1 with the Uniform Mapping technique had the longest run time at about 22 seconds. This informal performance test on a personal laptop illustrates the feasibility of the Comparison technique, as this is a non-exceptional amount of running time.

As with the Simulation time results, these numbers do not reflect the amount of time it would take a person to analyze the output of Comparison. Additionally, the results do not consider (since this experiment did not consider it) how long it would take to retrieve a relevant alternative model. Nonetheless, the time results provide some insight into the feasibility of a Comparison implementation.

5.2.2.2 Q2: Number of Invalid Differences

The results of each trial run were manually analyzed to determine the number of invalid model differences. An invalid difference is one that involves neither the parts of the source model that would be intentionally changed (if any) in the target model due to that model's configuration or the parts of the target model that were changed. Tables 5.11 and 5.12 present the results of this analysis. The exact number of differences is not necessarily worthy of detailed consideration here because, for example, an unmapped model element like a transition that may contain one or more sub-elements (explanations in this example) will still show in the results as a single difference. Fortunately, the results in the aforementioned tables are stark enough that the exact number of invalid differences need not be considered when comparing Compositional Mapping configurations against Uniform Mapping.

In trials 1-3 and 7-8, all three technique configurations found no invalid differences. Indeed, in trials 1, 2, 7, and 8, the three configurations identified no differences between the two models. In trials 7 and 8, this is expected since the source and target models are identical to each other, so this is evidence of baseline good functionality in the approaches. Similarly, the models in trials 1 and 2 only differ from each other in that some of their terminology has changed. The structure of the models (e.g., the number and composition of the behaviors, or the number of components) did not change. This makes sense for Uniform Mapping because it utilizes an implementation of the Structure-Mapping Engine (SME) [44], which ignores superficial differences. This also makes sense for Compositional Mapping, which (a) also utilizes an implementation of SME for its analogical subalgorithms and (b) uses a synthetic semantic comparison technique to also ignore the particular superficial differences introduced in this model configuration. Furthermore, the second step of Comparison, which utilizes the alignment results produced by Compositional Mapping or Uniform Mapping, also uses the same synthetic semantic comparison technique, also allowing it to ignore the

Table 5.11: Number of invalid differences detected by trial and technique configuration, trials 1 through 4

Trial	**Model and Model Configuration**	**Technique Configuration**	**# Invalid Differences***
1	Electric Toothbrush, Different Terminology	Compositional Mapping	0
		Compositional Mapping (No Interaction)	0
		Uniform Mapping	0
2	Owl Wing, Different Terminology	Compositional Mapping	0
		Compositional Mapping (No Interaction)	0
		Uniform Mapping	0
3	Friction Drag, Combined Components	Compositional Mapping	0
		Compositional Mapping (No Interaction)	0
		Uniform Mapping	0
4	Synthetic Car, Combined Components	Compositional Mapping	0
		Compositional Mapping (No Interaction)	0
		Uniform Mapping	5 (1 in S, 2 in F’s, 2 in B’s)

** When a non-zero value, the total is broken down into the number of differences in the Structure Submodel (S), Behaviors (B’s), and Functions (F’s)*

superficial differences in the Different Terminology model configuration.

Next, consider trials 3 through 6. All four of these trials represent situations where the structure of the model *did* change. Trials 3 and 4 cover the Combined Components model configuration, which was where two components and their attributes in the structure submodel were combined into a single component and single set of attributes, and this combination was reflected elsewhere in the model. In the case of both trials 3 and 4, this means changes in both the behaviors and functions. In trial 3, all three technique configurations found 0 invalid differences. However, in trial 4, both Compositional Mapping configurations found 0 invalid differences whereas Uniform Mapping found 5. Why was this the case?

Comparing the structure submodel results of both trials, one sees a potential reason.

Table 5.12: Number of invalid differences detected by trial and technique configuration, trials 5 through 8

Trial	Model and Model Configuration	Technique Configuration	# Invalid Differences*
5	Train, Behavior/Function Reorganization	Compositional Mapping	0
		Compositional Mapping (No Interaction)	0
		Uniform Mapping	3 (3 in B's)
6	Medical Patch, Behavior/Function Reorganization	Compositional Mapping	8 (8 in B's)
		Compositional Mapping (No Interaction)	8 (8 in B's)
		Uniform Mapping	0
7	Owl Wing, Original	Compositional Mapping	0
		Compositional Mapping (No Interaction)	0
		Uniform Mapping	0
8	Synthetic Car, Original	Compositional Mapping	0
		Compositional Mapping (No Interaction)	0
		Uniform Mapping	0

* *When a non-zero value, the total is broken down into the number of differences in the Structure Submodel (S), Behaviors (B's), and Functions (F's)*

In trial 3, all three technique configurations detected differences related only to those components and attributes that were changed or (if the source) would be changed. However, in trial 4, while the Compositional Mapping technique configurations continued this pattern, Uniform Mapping made a map involving an unchanged component, causing an unchanged component in the other model to go unmapped. Although one cannot directly, causally link this difference to the outputs, it makes sense. Upon reviewing the invalid differences in Uniform Mapping's results, all of the differences relate either directly or indirectly to this mismapped component in either the source or target model. Uniform Mapping must have gotten confused by the structural change in the models, leading to invalid differences detected, whereas Compositional Mapping did not.

Perhaps the reason that neither Compositional Mapping technique configuration fell to this confusion is due to the way in which Compositional Mapping decomposes the alignment problem. Analogical mapping for the structure submodel in Compositional Mapping (both configurations) is done first and is isolated from the other parts of the model. Thus, the information presented to the Structure-Mapping Engine (via Case Mapper) is different. This may have led the reasoner to decide to map things differently.

Trials 5 and 6 cover the Behavior/Function Reorganization model configuration. This model configuration also involved structural change in the models. In Behavior/Function Reorganization, a subfunction and its associated behavior were integrated into the superfunction and the superbehavior. Note that both configurations of Compositional Mapping had identical results for these trials because nothing was manually aligned at the interaction prompts, so these results will be treated as if they were one configuration for simplicity. In trial 5, Compositional Mapping with its zero invalid differences did better than Uniform Mapping with its 3. However, in trial 6, the situation was flipped: Compositional Mapping had 8 invalid differences compared to Uniform Mapping, which had zero.

Reviewing the results and analysis, these trials illustrate an interesting difference between the Compositional and Uniform Mapping approaches. Compositional Mapping decomposes the alignment problem into subproblems and solves each in isolation (plus constraints), so it will never, for example, look outside of a mapped pair of behaviors when trying to map states within a behavior. By contrast, Uniform Mapping attempts to align both models (conceptually) all at once, giving it the flexibility of looking anywhere in the model for alignment. This is especially meaningful in the case of Behavior/Function Reorganization because parts of the changed superbehavior and superfunction literally come from a second function and behavior in the other model.

Trials 5 and 6 show that this difference can be both a strength and a weakness. In trial 5, Uniform Mapping maps some states and transitions across behaviors. It does successfully (and smartly) map some new states and transitions in the superbehavior to the subbehavior counterparts in the other model. Unfortunately, cross-behavior mapping also leads to it mapping two transitions not involved in the model configuration across behaviors and one unrelated transition going unmapped. It is this confusion that leads to the invalid differences detected in this technique configuration. Compositional Mapping, which literally cannot do such cross-behavior mapping, does not suffer from this issue.

Trial 6, however, shows the other potential in action. In it, Uniform Mapping still does cross-behavior alignment (and does it well!), and it does so without introducing any invalid differences. However, Compositional Mapping gets confused and mismaps states and transitions (and thus their conditions and explanations) within a mapped behavior. This may be due to the changed behavior having a different number of states and transitions than the other behavior, opening the door for such confusion. Thus, because Compositional Mapping was unable to look outside of the mapped behavior pair, it failed to reconcile these differences, leading to invalid differences. This did happen somewhat in trial 5's results as well, but there Compositional Mapping did not raise any invalid differences due to it.

How exactly to optimize the ability to do (or not to do) cross-behavior mapping–or, more generally, cross-model element mapping–with an eye to avoid invalid differences remains an area for future research. A good starting point would be to identify what differs between the models in trials 5 and 6 that causes these split results between the two strategies. This starting point could be expanded to also consider the differences in results during trials 3 and 4 as well: what was different about the models in trial 4 that triggered the confusion seen in Uniform Mapping when trial 3 did not?

5.2.2.3 Q3: Which Technique Configuration was Superior?

Given the results of the previous question, Table 5.13 synthesizes these results into which technique configuration was superior per trial. A technique configuration is superior to the others if it has fewer invalid model differences. The analysis found that overall, the two Compositional Mapping technique configurations (i.e., Compositional Mapping and

Compositional Mapping (No Interaction)) were superior to Uniform Mapping. The two Compositional Mapping technique configurations were superior in two trials (4 and 5), whereas the Uniform Mapping technique configuration was only superior to them in one trial. In the other four trials, the configurations were tied, with all three of them having no invalid differences in those trials.

Table 5.13: Which technique configuration was superior per trial

Trial	Model Configuration	Superior Technique Configuration
1	Different Terminology	None (all were perfect)
2		None (all were perfect)
3	Combined Components	None (all were perfect)
4		Compositional Mapping (both versions were equal)
5	Behavior/Function Reorganization	Compositional Mapping (both versions were equal)
6		Uniform Mapping
7	Original	None (all were perfect)
8		None (all were perfect)

These results are positive in favor of Compositional Mapping, providing partial support of its superiority over Uniform Mapping for the purposes of comparison-based evaluation. However, they also suggest room for improvement. All technique configurations were very good in cases where the structure of the model (note: this phrase should not be confused with SBF*'s structure submodel) did not change (Original and Different Terminology model configurations), but they start to break down when the structure does change (Combined Components and Behavior/Function Reorganization). Given that Uniform Mapping was better than Compositional Mapping in one of the Behavior/Function Reorganization trials, this kind of structural change should be prioritized for future work.

Furthermore, the analysis technique for this dissertation only attempts to address a baseline criterion for good results: did they reflect things that actually changed or, for the source models, were going to be changed? This does not address if the results were useful to human reasoners or if they reflected reasonable alignment decisions. For example, both the Compositional Mapping technique configurations mismapped two component attributes in trial 4. These attributes fell within the scope of valid differences and so did not detract from Compositional Mapping's score, but they nevertheless were poor alignment decisions. Future research should conduct a deeper analysis of Comparison's output along these lines to determine the extent to which the results were good results and then how to improve the reasoning process to better those results.

5.3 Conclusions and Takeaways

This final section of the chapter discusses to what extent the relevant hypotheses were confirmed or rejected based on the Computational Experimentation.

5.3.1 Reflection on Research Questions and Hypotheses

The Computational Experimentation addresses Hypotheses 2.1, 2.2, 3, and 4 and their related Research Questions. It also indirectly tests part of Hypothesis 1 and its Research Question. Each will be addressed in turn, with Hypothesis 1 addressed last.

- **Research Question 2.** How might an AI Agent check a design model for internal consistency?

  Hypothesis 2.1. It may do so by simulating the model and using the results to check the process and functional claims made by the model. The Simulation technique of DESC illustrates this hypothesis.

  Hypothesis 2.2. The SBF* functional modeling language provides the syntax and semantics needed for automated simulation.

These hypotheses were **supported** by the results of this Computational Experimentation.

The experimentation results show the sufficiency of Simulation's features for checking the internal consistency of a model, specifically with regards to the class of modeling problems captured by the various model configurations. The Complete configuration always identifies all valid issues when any exist and never identifies and invalid issue. Additionally, the results also show the necessity of these features for the class of problems because (a) the technique configuration without implicit value forwarding fails to run to completion in most cases and (b) there is at least one trial for each of the other two ablated technique configurations where it either fails to correctly identify all valid issues or identifies one or more invalid issues. Thus, Hypothesis 2.1 is supported in light of these findings. Additionally, all of these trials operated over the SBF* knowledge representation that illustrates a representation with the syntax and semantics for automated simulation. This supports Hypothesis 2.2, for its features supported automated simulation.

- **Research Question 3.** How might an AI agent check a design model for external consistency?

  Hypothesis 3. It may do so by analogically comparing a given model against another model of the same design, where differences represent potential issues. The Comparison technique of DESC illustrates this hypothesis.

This hypothesis was also **supported** by the results of the Computational Experimentation. Across the board the number of invalid differences found by the technique configurations were either zero or in the single digits. This is a reasonably small number. (Granted, some differences may hide others.)

To justify that claim, consider the following. Trial 4 was one trial that had invalid differences. The Synthetic Car model in the Combined Components configuration has 4 components with 6 total attributes in its structure submodel; 3 functions with 6 total conditions in them; and 3 behaviors with 12 total states, 23 total conditions within those states, 9 total transitions, and 9 total explanations within those transitions. The 5 invalid differences detected by Uniform Mapping in this circumstance is small compared to the size of one of the models involved in the comparison. Consider the behaviors (where the invalid differences occurred) in the configured models for Trials 5 and 6. In the Train model in the Behavior/Function Reorganization configuration, there are 4 behaviors with 14 total states, 31 total conditions within those states, 10 total transitions, and 18 total explanations in those transitions. The 3 invalid differences detected by Uniform Mapping again is small compared to this size. The Medical Patch model in the Behavior/Function Reorganization configuration has 2 behaviors with 9 total states, 27 total conditions within those states, 7 total transitions, and 14 total explanations within those transitions. Again, although 8 invalid differences is a higher number than the other invalid difference totals, this is small compared to the size of all the behaviors in one of the two models in the comparison.

Therefore, because the number of invalid differences detected was small when they occurred and because invalid difference detection trending towards zero means that Comparison succeeds at mostly detecting only those differences relevant to the change that occurred in the models (either model elements that were changed or were going to be changed), Hypothesis 3 is supported in that the Comparison technique would be decent as a method for external consistency checking. Granted, there is room for improvement and for further study to determine what valid differences would be helpful (salient) to detect for this purpose.

- **Research Question 4.** Analogical comparison implies mapping. How might an AI agent map two models?

  Hypothesis 4. It may do so by decomposing the mapping task, solving the subtasks, and using solutions from earlier subtasks to constrain and inform solutions to future subtasks. Compositional Mapping illustrates this hypothesis.

Implicit in this hypothesis is that the ideas presented herein (and illustrated by Compositional Mapping) are superior to a mapping technique without them.

This hypothesis was **partially supported** by the results of the Computational Experimentation. The previous discussion on the support for Comparison also provides evidence for Compositional Mapping's efficacy since it was part of the Comparison trials, so the question really becomes whether Compositional Mapping as a theory of analogical mapping is superior to a mapping algorithm without its features. Such a mapping technique is embodied by Uniform Mapping. The results (number of invalid differences) showed that Compositional Mapping with and without interactive subalgorithms is superior to Uniform Mapping in 2 trials, whereas Uniform Mapping only bests either Compositional Mapping configuration on 1 trial. Although Compositional Mapping is superior, that it only narrowly bests Uniform Mapping leads Hypothesis 4 to be given only partial support.

- **Research Question 1.** How might an AI agent support systematic and repeatable incremental design revision?

  Hypothesis 1. It may do so through checking the internal and external consistency of design models.

Because the hypotheses for internal consistency and external consistency checking were both supported, the aspect of this hypothesis that relates to an AI agent being capable of checking for internal and external consistency issues is **supported**. Note that the Computational Experimentation does not test the extent to which such checking will lead to incremental model revision. However, this is explored in the Usefulness Study in Chapter 6.

5.3.2 On Feasibility

Finally, although only a very limited, informal performance evaluation was done via collecting run times, these results support the feasibility of Simulation and Comparison implementations in actual use. Both techniques showed trials whose mean and median times lasted well under one minute. This implies that users of these techniques would not have long to wait for feedback from the computer about their models. While not taking into account the amount of time it takes for a person to cognitively process the feedback given by DESC or to develop the models given as input, this nevertheless provides promising results for the use of DESC in the broader hypothesized incremental design revision cycle since its computation will not present a bottleneck.

```
PatchResistsPullout is a function
  It is denoted by the verb "Resist"
  It provides: (
    Patch.Attached = "True"
  )
  It is achieved by the behavior PatchResistsPulloutBehavior

TipsResistPullout is a function
  It is denoted by the verb "Resist"
  It provides: (
    ConicalTips.Inserted = "True"
  )
  It is achieved by the behavior TipsResistPulloutBehavior

TipsSwell is a function
  It is denoted by the verb "Grow"
  It provides: (
    ConicalTips.FrictionAmount = "10.0"
  )
  It is achieved by the behavior TipsSwellBehavior
```

Snippet of the Original configuration of the Medical Patch model

```
PatchResistsPullout is a function
  It is denoted by the verb "Resist"
  It provides: (
    Patch.Attached = "True"
    and ConicalTips.FrictionAmount = "10.0"
  )
  It is achieved by the behavior PatchResistsPulloutBehavior

TipsResistPullout is a function
  It is denoted by the verb "Resist"
  It provides: (
    ConicalTips.Inserted = "True"
  )
  It is achieved by the behavior TipsResistPulloutBehavior
```

Snippet of the Behavior/Function Reorganization configuration of the Medical Patch model. Meaningful changes from the Original configuration are in bold

Figure 5.3: Example for the Behavior/Function Reorganization model configuration from the function perspective

```
T2 is a transition from State2 to State3
  It has the explanation: (
    the function TipsSwell
  )
```

Snippet of the Original configuration of the Medical Patch model

```
Sub_T1 is a transition from State2 to Sub_State2
  It has the explanation: (
    the equation TipsStartSwelling, described as "eq:
    ConicalTips.Size is directly proportional to 1.0"
    and the equation LargerTipsMeansMoreResistanceForce, described
    as "eq: ConicalTips.FrictionAmount =
    ConicalTips.FrictionAmount:Before + 5.0"
  )

Sub_State2 is a state
  It has the condition: (
    ConicalTips.Size = "Medium"
    and ConicalTips.FrictionAmount = "5.0"
  )

Sub_T2 is a transition from Sub_State2 to State3
  It has the explanation: (
    the equation TipsFinishSwelling, described as "eq:
    ConicalTips.Size is directly proportional to 1.0"
    and the equation LargerTipsMeansMoreResistanceForce, described
    as "eq: ConicalTips.FrictionAmount =
    ConicalTips.FrictionAmount:Before + 5.0"
  )
```

Snippet of the Behavior/Function Reorganization configuration of the Medical Patch model

Figure 5.4: Example for the Behavior/Function Reorganization model configuration from the behavior perspective

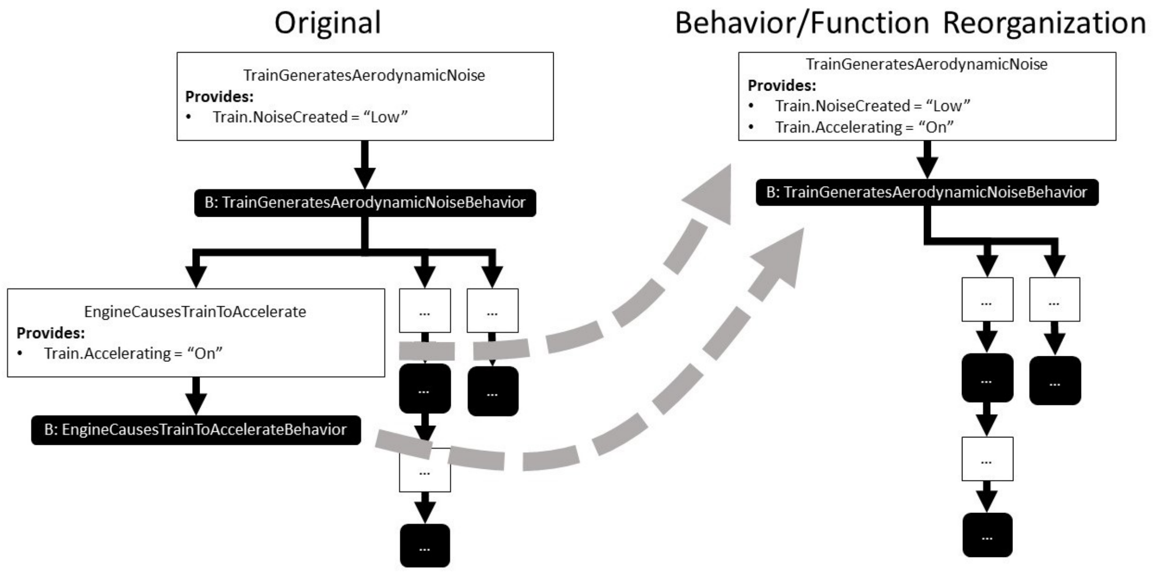

Original
Behavior/Function Reorganization
TrainGeneratesAerodynamicNoise
Provides:
• Train.NoiseCreated = "Low"
B: TrainGeneratesAerodynamicNoiseBehavior
EngineCausesTrainToAccelerate
Provides:
• Train.Accelerating = "On"
B: EngineCausesTrainToAccelerateBehavior
TrainGeneratesAerodynamicNoise
Provides:
• Train.NoiseCreated = "Low"
• Train.Accelerating = "On"
B: TrainGeneratesAerodynamicNoiseBehavior

Figure 5.5: Diagrammatic visualization of the Behavior/Function Reorganization configuration for the Train model. A behavior/function pair is integrated into their superbehavior/-superfunction

```
Transition_1 is a transition from Start_State to State_2
  It has the explanation (
    the equation Brush_Head_Rotates_to_New_Orientation, described
    as "eq: Brush_Head.Clockwise_Rotation_Orientation_in_Degrees =
    Brush_Head.Clockwise_Rotation_Orientation_in_Degrees:Before +
    90.0"
    and the equation
    Available_Intermittent_Rotational_Movement_is_Used_to_Rotate_
    Brush_Head, described as "eq:
    Cam_and_Gears.Rotation_Available_in_Degrees =
    Cam_and_Gears.Rotation_Available_in_Degrees:Before - 90.0"
  )
```

Snippet of the Original configuration of the Electric Toothbrush model

```
Transition_1 is a transition from Start_State to State_2
  It has the explanation (
    the equation Brush_Head_Rotates_to_New_Orientation, described
    as "eq: Brush_Head.Clockwise_Rotation_Orientation_in_Degrees =
    Brush_Head.Clockwise_Rotation_Orientation_in_Degrees:Before +
    45.0"
    and the equation
    Available_Intermittent_Rotational_Movement_is_Used_to_Rotate_
    Brush_Head, described as "eq:
    Cam_and_Gears.Rotation_Available_in_Degrees =
    Cam_and_Gears.Rotation_Available_in_Degrees:Before - 90.0"
  )
```

Snippet of the Wrong Equation configuration of the Electric Toothbrush model. Meaningful changes from the Original configuration are in bold

Figure 5.6: Example for the Wrong Equation model configuration

```
State-3 is a state
  It has the condition: (
    Wheels.RPM = "10.0"
    and Car.Velocity = "10.0"
  )
```

Snippet of the Original configuration of the Synthetic Car model

```
State-3 is a state
  It has the condition: (
    Wheels.RPM = "20.0"
    and Car.Velocity = "10.0"
  )
```

Snippet of the Wrong State Value configuration of the Synthetic Car model. Meaningful changes from the Original configuration are in bold

Figure 5.7: Example for the Wrong State Value model configuration

```
TipsSwell is a function
  It is denoted by the verb "Grow"
  It provides: (
    ConicalTips.FrictionAmount = "10.0"
  )
  It is achieved by the behavior TipsSwellBehavior
```

Snippet of the Original configuration of the Medical Patch model

```
TipsSwell is a function
  It is denoted by the verb "Grow"
  It provides: (
    ConicalTips.FrictionAmount = "20.0"
  )
  It is achieved by the behavior TipsSwellBehavior
```

Snippet of the Wrong Value in a Function Provides Condition configuration of the Medical Patch model. Meaningful changes from the Original configuration are in bold

Figure 5.8: Example for the Wrong Value in a Function Provides Condition model configuration

Non-contradictory modeling decisions: Two models of the same design differ in ways such that their details are not in direct contradiction of each other.

- **Different function decomposition:** The model has a different number of function/behavior pairs because the modeler decomposed the functions of the system are differently. **This is covered by the *Behavior/Function Reorganization* model configuration.**
- **Different structure decomposition:** The model has a different number of components (and potentially different number of attributes) because the modeler decomposed the structure of the system differently. **This is covered by the *Combined Components* model configuration.**
- **Different terminology:** The model has different names for model elements. **This is covered by the *Different Terminology* model configuration.**
- **Less/more information:** A given model is either a subset or a superset of another model, reflecting more or less detail given in the model.
- **Different attribute types:** One or more attributes have different types, perhaps reflecting a decision to go with qualitative over quantitative reasoning.

Contradictory differences: Two models of the same design differ in ways such that their details are in direct contradiction of each other.

- **Behavioral differences:** One or more behaviors of the model differs from the other model in such a way that they do not capture the same process account. For example, meaningfully different explanations are given, or values are not compatible even accounting for attribute type differences.
- **Functional differences:** One or more functions of the model differ from the other model in such a way that they do not capture the same functions. In SBF* as it appears in this dissertation, this could be due to different component attribute values in provides conditions that cannot be accounted for by attribute type differences and/or a non-sub/super set of provides conditions relative to the other model.
- **Structural differences:** One or more components of the model or the set of connections differ from the other model in such a way that the structure submodels depict conflicting structures of the design. For example, the sets of components and/or attributes are not a sub- or superset of the components/attributes in the other model nor do they reflect a different decomposition.

Figure 5.9: Speculated taxonomy of SBF* model differences

Syntax: The syntax of the modeling language is incorrect.

> **Equation:** The syntax of an equation in an explanation is incorrect.
>
> **Anywhere else:** The syntax elsewhere in the model is incorrect.

Cross-referencing: A reference is incorrectly made in the model to an element elsewhere in the model.

> **Non-existent:** A reference is made to a non-existing model element.

Function satisfying: One or more of a function's provides conditions is not achieved by the function's associated behavior.

> **Incorrect attribute value:** The associated behavior's output does not achieve one of its provides conditions. **This is covered by the *Wrong Value in a Function Provides Condition* model configuration.**
>
> **Missing attribute value:** The associated behavior's output does not contain an attribute value expected to be set by a function's provides conditions.

Figure 5.10: Speculated taxonomy of SBF* model defects, part 1

Behavior consistency: Given two states and a transition between them, one or more state conditions in the later state does not logically follow by starting with the earlier state's conditions and tracing the impact of the transition explanations.

Incorrect condition: One or more conditions are assigned blame for this defect. **This is covered by the *Wrong State Value* model configuration.**

Incorrect explanation: One or more explanations are assigned blame for this defect. **This is covered by the *Wrong Equation* model configuration.**

Behavior semantics: Covers other semantic defects in behaviors besides consistency issues.

Unsolvable equations: One or more equations in explanations has references on the right-hand side that are not resolvable to values given their chronological position within their behavior.

Function subfunction infinite loops: The function-subfunction decomposition created by tracing function explanations in behaviors forms at least one cycle of any length. This is semantically nonsensical.

Structure submodel semantics: An aspect of the structure submodel violates the semantics of the modeling language.

Self-referential connections: A connection is between one component and that same component, both with the same connecting point. This is nonsensical and thus a defect.

Missing quantity space: A Qualitative-type component attribute does not have a quantity space declared as its description. This prohibits qualitative reasoning involving this attribute because no quantities can be referenced or reasoned about.

Figure 5.11: Speculated taxonomy of SBF* model defects, part 2

I ran a simulation to verify your functions and behaviors. Below, I point out any issues that I identified with them.
Functions

- For Move_Head_Clockwise:
  - Your function said Brush_Head.Clockwise_Rotation_Orientation_in_Degrees = 90.0. However, based on your associated behavior, I think that it should be 45.0.

Behaviors

- For Behavior_for_Clean_Teeth:
  - For state State_5:
    - You said Brush_Head.Clockwise_Rotation_Orientation_in_Degrees = 90.0. I think that it should be 45.0.
  - For state State_6:
    - You said Brush_Head.Clockwise_Rotation_Orientation_in_Degrees = 0.0. I think that it should be -45.0.
- For Behavior_for_Move_Head_Clockwise:
  - For state State_2:
    - You said Brush_Head.Clockwise_Rotation_Orientation_in_Degrees = 90.0. I think that it should be 45.0.

Figure 5.12: Sample Simulation results

I ran a simulation to verify your functions and behaviors. Below, I point out any issues that I identified with them.
I identified zero issues with your model. Good job!

Figure 5.13: Sample Simulation output when it finds no issues

```
Error: runNoImplicitValueForwardingTrial, caught sim exception: Could not process
all equations in transition T2 in behavior EngineCausesTrainToAccelerateBehavior,
probably due to incorrect syntax or one or more unresolved attributes in the expression.
```

Figure 5.14: Sample error message from Simulation

CHAPTER VI

USEFULNESS STUDY

This chapter discusses a pilot study with human participants that tested the usefulness of the Design Evaluation through Simulation and Comparison (DESC) computational technique. It describes the purpose, design, results, and analysis of the study.

6.1 Purpose

In the Usefulness Study, participants were tasked to complete a partially constructed model related to either a technological system (an electric toothbrush) or a physical phenomenon relevant to a biological system (friction drag). Some participants were given access to an implementation of DESC (referred to in this experiment as the automated verification technology(ies)) to help them in their modeling exercise, whereas others were not. (Note that this was an earlier implementation than that described in Chapters 4 and 5.)

To scope this experiment, participants conducted their task outside of a design context. However, future work could incorporate such a context to make the experiment more authentic to design and to provide participants with greater motivation to understand the concept being modeled.

The Usefulness Study explores the part of Hypothesis 1 (implied in this hypothesis) and its associated Research Question that relate to the AI agent supporting incremental design revision. The hypothesis and research question are as follows:

- **Research Question 1.** How might an AI agent support systematic and repeatable incremental design revision?

  Hypothesis 1. It may do so through checking the internal and external consistency of design models.

Relative to the Computational Experimentation in Chapter 5, the Usefulness Study serves a different purpose. The Computational Experimentation shows the extent to which an implementation of DESC can evaluate the internal and external consistency of models. However, the Experimentation does not address DESC's impact on human reasoners, which is important because it gets to DESC's ability to support incremental design revision. By contrast, this Usefulness Study does explore to what extent can the AI agent (DESC) can support incremental design revision by observing DESC's impact on human reasoners.

Due to the way the Usefulness Study was conducted and the results analyzed, the hypothesized incremental design revision cycle was not directly observed. Instead, the final model and final version (plus the initial version to use as contrast) of a participant's understanding were analyzed, which are the outcomes of any revising that was done during the experiment. Thus, this experiment *explores* the topic of an AI agent supporting incremental design revision, but more research is warranted to analyze it in detail.

6.2 Description

The Usefulness Study was a controlled, between-subjects study. It had the following independent and dependent variables.

6.2.1 Independent Variables

- Whether a participant had access to and training about DESC's implementation for model-building (Experimental condition) or not (Control condition).
- Whether a participant was prompted and given information to construct a model of friction drag (Biological condition) or of an electric toothbrush (Technological condition).

These conditions form a 2x2 matrix of all the conditions in the experiment. Table 6.1 visualizes this matrix. When only a single dimension of this matrix is referred to (e.g., the Experimental condition) without the other, it includes participants in both of the second type of condition (e.g., the Experimental condition refers to participants in both the *Experimental and Biological* and *Experimental and Technological* conditions).

Table 6.1: Matrix of participant conditions in the Usefulness Study

	Biological	**Technological**
Experimental	Experimental and Biological	Experimental and Technological
Control	Control and Biological	Control and Technological

The ordering of the conditions was randomly generated except that each group of four assignments had to fulfill all four conditions. For example, if the first assignment was randomly selected to be *Control and Biological* then the second assignment was randomly selected to be either *Experimental and Biological*, *Experimental and Technological*, or *Control and Technological*. This meant that the fourth assignment was always the last remaining condition in the set of four rather than being randomly picked. This assignment strategy ensured an even distribution of conditions across the participants. Because P1 was excluded (see Section 6.2.6 below), their condition was treated as if it had not happened when constructing assignments for P2 and onwards.

Participants were assigned to a condition based on the order in which they actually participated in the experiment. When participants had to reschedule, rescheduling decisions were made without considering how this would change the participant to condition assignment.

6.2.2 Dependent Variables

The experiment also had the following dependent variables.

- Participant demographics.
- Survey results.
- The model that each participant constructed.

6.2.3 Experimental Procedure

Each participant in the study underwent the following procedure.

1. Informed consent was gained.

2. (Demographics) Participants took the Initial Survey, which collected demographic information.

3. (Training) Participant was trained on Structure-Behavior-Function modeling, how to read the modeling interface website and how to construct behaviors within it, and how to write formulas (equations). If the participant was in the Experimental condition, they were also trained on the automated verification technologies.

4. (Pre-Test) Participant read the appropriate Modeling Prompt for their condition and then took the Pre-Modeling Survey for their condition.

5. (Model-building Preparation) Participant was given the appropriate Source Materials and Set of Expected Function-Subfunction Relationships. They were given access to the modeling interface with the appropriate model preloaded into it. Here, the Structure and Function models were filled out and unchangeable. Relatively empty behaviors (containing only an empty Start State, an empty Stop State, and an empty Transition between them) were also provided. Participants were only able to work within the given behaviors; they could not add or delete whole behaviors.

6. (Model-building) Participants were told that they had up to 45 minutes to build their model. They would be stopped at 45 minutes but could end early if they chose to do so. They were to use the Source Material as their main source of information but could go beyond it if they knew more or disagreed with it. They also had access to the Formula Writing Guide, the Modeling Prompt, and the Set of Expected Function-Subfunction Relationships. If the participant was in the Experimental condition, they

were also told to write down any circumstances where they intentionally disregarded or ignored the automated verification technologies' advice and why. (See Section 6.5). Participants were not allowed to use anything else to work with during their modeling besides the modeling interface. Participants were allowed to ask questions of the person running the experiment (which in all cases was the dissertation author) if they wished, who responded to all questions when asked but intentionally did not provide answers to questions regarding the content of the model being built.

7. (Post-Test) Once model-building was completed, the model being built was closed and the participant only kept access to the Modeling Prompt and Formula Writing Guide. They were then asked to complete the Post-Modeling Survey appropriate for their condition, which was identical to the Pre-Modeling Survey.

8. (Exit Survey) The participant then lost access to all materials and was asked to complete the Exit Survey. Participants in the Experimental condition had more questions on their exit survey, corresponding to topics related to the automated verification technologies. Note that there was accidentally a one word difference (technology versus interface) in the first question between the Control and Experimental conditions instead of them being identical as intended.

See Appendix C for re-creations of the experiment materials. Note that the source materials for model building are given brief descriptions and citations rather than being re-created since they were taken from others' work.

6.2.4 Configurations of Functional Model Simulation and Comparison

Participants in the Experimental condition had access to an implementation of DESC, which was referred to as the automated verification technologies. Due to when the Usefulness Study occurred, this was an earlier version of DESC compared to that described and tested in Chapters 4 and 5. Nevertheless, the version that the participants had access to in this experiment was conceptually similar: the Simulation technique evaluated the participant's model through simulation and Comparison did so by comparing it against another model. Furthermore, based on records, the participants should have used the same version of the SBF* modeling language as used in the Computational Experimentation. Thus, the results of the Usefulness Study remain relevant to the theoretical questions asked in this dissertation.

The implementations of the Simulation and Comparison techniques of DESC had particular configurations in the Usefulness Study. Simulation was implemented with implicit value forwarding, function reasoning, and reasoning with equations in explanations all enabled. This configuration was intended to enable Simulation to work as well as possible. That said, participants were not taught about qualitative equations for the sake of simplicity, so that aspect of Simulation was effectively disabled for the experiment.

Comparison utilized Compositional Mapping, with the top-level ordering of subproblems being Structure model, Function model, then Behavior model. All of the subproblem algorithms were heuristic-based algorithms, with the behavior subalgorithm being a particularly advanced one that traversed behaviors. This configuration was used for two reasons. First, heuristic algorithms were used instead of analogical ones because (although this appears to have been a fluke) the implementation of the analogical algorithms was quite slow at the time and (as a practical reason) time ran out to set up analogical algorithm functionality. Second, the Usefulness Study is constrained such that a participant's model will always have an identical Structure submodel and Functions to the model it gets compared against, so more sophisticated algorithms beyond heuristic algorithms were unnecessary to map those parts.

6.2.5 Gold Standard Models

With the help of two domain experts, a gold standard model was created for each of the Biological and Technological conditions. To develop the gold standard models, a series of meetings were conducted separately with two domain experts, Dr. Julie Linsey and Dr. Marc Weissburg (both from Georgia Tech), with the intent of developing expert versions of the models for the Technological and Biological conditions, respectively. The content of these meetings varied and the modeling topics evolved, but this work eventually resulted in the dissertation author constructing a model (or finishing the construction of a model, in the case of the work with Dr. Linsey) of what are now the Technological and Biological modeling topics. This model was then presented (in an accessible form) to the appropriate expert for review. Afterwards, the model was revised based on their feedback and on any additional appropriate changes. These revised versions were used in the experiment. These models are present–except with their names changed in the text of the model–in Appendix D under the code names Friction Drag (for the Biological condition) and Electric Toothbrush (for Technological)[1].

A gold standard model in this experiment serves multiple purposes: (a) it acts as the perfect or correct model of the topic in question for analysis purposes, (b) parts of it were used as the starting place for a participant's model, and (c) they populated the implementation's knowledgebase and were used by the Comparison technique to derive comparisons with participant models.

6.2.6 Preparation for Usefulness Study

Two informal tests with one person each were ran in preparation for the Usefulness Study to evaluate the per-participant procedure of this experiment. The goal was to identify if the training was the appropriate length relative to experimental time constraints and if the experiment flowed as expected. As a result of these tests and follow-up discussion with colleagues, the training was reformulated, modeling was simplified, and the experiment was overall more polished. Next, another test was conducted with the second person used in the earlier tests specifically to address usability of the modeling interface, resulting in

[1]Due to organizing reasons, there is a small possibility that these models are not identical to those used in the Usefulness Study. However, it is believed that they are.

more polish.

Additionally, the first participant of the experiment, P1, acted as an unintentional additional test. P1 encountered a severe error when they created a function-subfunction infinite loop (defined below in Operationalization 4, Measure 2). After P1's trial, a preventative check for this error was added to the modeling interface. Simulation's error reporting was also improved, responding to feedback from both P1 and from the second informal test helper. Thus, P1's experience led to improvements in the experiment. The cost was that P1's data is not reported below because they received a different treatment (used a different version of the modeling interface and automated verification technologies) than the other participants.

6.3 Results Preface

The following sections present the results and analysis of the models that participants constructed and the surveys that they filled out. First, some preliminary information.

6.3.1 Participants

Table 6.2 presents the conditions of the experiment's participants.

Table 6.2: Participants and conditions

Participant	**Condition**
P1	Experimental and Biological
P2	Experimental and Technological
P3	Control and Biological
P4	Control and Technological
P5	Experimental and Biological
P6	Experimental and Biological
P7	Experimental and Technological
P8	Control and Technological
P9	Control and Biological

Note two things. First, P1 is excluded from the below results for reasons identified above. Second, P7 has been retroactively re-categorized as a Control condition participant despite being in the Experimental group. This is because P7 did not use the automated verification technologies despite them being available. Thus, this participant was effectively a Control condition participant.

Given these notes, the sample sizes for the conditions can be seen in Table 6.3. The results will note when these sample sizes differ due to particular circumstances.

Table 6.3: Sample sizes by condition after P1 exclusion and P7 recategorization

	Control	**Experimental**
Biological	n = 2	n = 2
Technological	n = 3	n = 1

6.3.2 Statistical Significance Calculation

Note that p-values were calculated for many of the results below, using an independent samples, two-tailed t-test. In no circumstance were the p-values calculated for these experimental results found to be statistically significant. Thus, the p-values are not shown hereafter and none of the analysis will rely on statistical significance as part of its argument.

6.4 Models

Each participant constructed a model during the experiment. This section analyzes those models. Specifically, this dissertation inspects only the last snapshot of each constructed model that was saved according to timestamps for simplicity. The critical question in this analysis is how to determine whether one model was better than another; essential to get at if Control or Experimental condition participants developed better models. Four operationalizations were created to determining this, each phrased as a hypothesis and representing a different perspective one might take on model quality. Each operationalization then has one or more measures, also phrased as hypotheses, that were applied to each model. The measures can be viewed as more detailed versions of the operationalizations.

The analysis done for the models in the Usefulness Study occurred over a long period of time, thus (for example) analysis that utilized the DESC implementation was done with the final versions (that which is discussed in Chapters 4 and 5) whereas the Usefulness Study was conducted with an earlier version of the implementation. Also, there exists a small possibility (due to organizing issues) that variants of the gold standard models may have been used in the analysis. Nevertheless, for any given measure, the same gold standard model version should have been used across all participants in that measure.

For reasons related to the trajectory of this dissertation research, the operationalizations here were not created with internal and external consistency in mind. However, in the descriptions of the operationalizations, whether or not they can be viewed as addressing these ideas is commented on. Regardless, the multiple operationalizations used to analyze models give a diverse perspective on the goodness of a model.

6.4.1 Operationalization 1

Experimental condition models will be more similar to the gold standard model than will the Control condition models.

This operationalization assumes that the gold standard model represents the perfect or correct model of the topic being modeled. Given this assumption, the closer a model comes to matching the gold standard model, the better it is. The following measures are thus three ways to investigate the amount of difference between a participant model and the related gold standard model.

Relating this back to questions of external and internal consistency, this operationalization relates to external consistency. The gold standard models act as an accurate representation of a concept that a given model should mirror, so by comparing (in various ways) a given model against its associated gold standard model, this operationalization can be viewed as checking the external consistency of the model. Measure 3's look at complexity has a somewhat tenuous relationship to this idea, but it is still essentially an act of comparison between the two models.

6.4.1.1 *Operationalization 1 - Measure 1*

Experimental condition models will more closely match the gold standard model than will the Control condition models.

This measure was calculated by comparing the participant's model against the relevant gold standard model using the final version of Comparison in the Compositional Mapping (No Interaction) technique configuration from Chapter 5. After running comparison, the number of identified differences was counted. Since the two models should be identical, the fewer the differences the better.

The number of differences is an imperfect tool for comparison between participants because (a) the computational experimentation shows that the Comparison technique is imperfect, (b) Comparison requires nearly exact equality between component attribute values and explanation descriptions (when they are used for reasoning) for them to be considered a non-difference, and (c) it will undercount the number of differences when states and transitions are unmapped by not showing any of the conditions or explanations, respectively, underneath them. Nevertheless, this approach provides an approximate, consistent measure of how different the two models are.

Results: The Experimental condition (n = 3) had an average of 50.33 differences. The Control condition (n = 5) had an average of 76.6 differences. Note that numerical values reported here and below may be rounded from their raw values for simplicity (e.g., 50.33 is rounded).

6.4.1.2 *Operationalization 1 - Measure 2*

The trajectory of attribute values in the Experimental condition models will more closely align with the gold standard model than will the Control condition models.

This measure focuses on how attribute values change in the behaviors. The change in attribute values represents a meaningful aspect of behavioral modeling, and this measure attempts to get at that perspective while avoiding other differences that might complicate or obscure comparison at this level of abstraction.

To calculate this measure, a Start value, Stop value, and Abstract, Directional (AD) value for each attribute on a per-behavior basis were determined for both the gold standard and participant models. For Start and Stop values represent only what was explicitly written in the states by the modeler or (see below) <`Undeclared-Q`> or <`Undeclared-D`> if no relevant value was present. AD values were derived from the Start and Stop values as described below.

The Start value for an attribute is that which appears in the Start state (first state) in the behavior. If there was no such value and the attribute appears elsewhere in the behavior (or appears in the matching behavior of the model against which this model is compared), its value was either <`Undeclared-Q`> or <`Undeclared-D`> depending on whether it was a Quantitative or Descriptive attribute, respectively.

The Stop value for an attribute is that which appears in the Stop state (final state) in the behavior. If no value appeared in the Stop state, the behavior was traversed backwards until finding the last declared value for that attribute, eventually repeating the Start state value if necessary. Again, if no such value was found (possible if the attribute only appears in the matching behavior of the model against which this model is compared), its value was either <`Undeclared-Q`> or <`Undeclared-D`> depending on the attribute type.

The Abstract, Directional (or AD) value represents another layer of abstraction. The numbers used in the models may not be tied to specific values in the source materials or elsewhere, so they may represent more general notions of quantities increasing or decreasing. Thus, differences in the precise values may not necessarily be meaningful and an abstraction (the AD value) may be useful. Given a Start and Stop value for a Quantitative attribute, the AD value for the attribute is either `Increasing-Q` if the Stop value is greater than the Start value, `Decreasing-Q` if the Stop value is less, `Static-Q` if they are equal, or `Unknown-Q` if either/both of the Start and Stop values were <`Undeclared-Q`>. Descriptive values are not numbers. However, it may still be valuable to look at how they change at an abstract level, so an abstract level was created for them an lumped in with AD values. They are classified as `Static-D` if both Start and Stop values were the same, `Different-D` if they were different, or `Unknown-D` if either/both of the Start and Stop values were <`Undeclared-D`>. (Note: there were no Qualitative attributes in the Usefulness Study models.)

Once the Start, Stop, and AD values were determined, the percent of these values in a participant's model that matched with the related gold standard model across all the behaviors was then calculated. A higher percentage is better because the percentage corresponds with how closely aligned the participant's model is with the gold standard model.

Results: Table 6.4 presents the results for this measure, which consist of the mean percent of matching values for Start, Stop, and Abstracted, Directional (AD) values.

Table 6.4: Operationalization 1, Measure 2 Results. Mean percent matching. (Higher is better)

Condition	Start Values	Stop Values	AD Values
Control (n = 5)	26.78	13.99	29.66
Experimental (n = 3)	35.27	22.64	40.23

6.4.1.3 Operationalization 1 - Measure 3

The complexity of the Experimental condition models will be closer to that of the gold standard model than will the Control condition models.

Another way to abstractly compare a participant model against the gold standard model is to look at their level of complexity. To do so, the number of explanations, conditions, formulas (equations), and states in both the gold standard models and the participant models were counted. Anything in the formula field counted as a formula. Then the difference (as an absolute value) was calculated for the number of each model aspect (e.g., explanations) between the participant's model and the related gold standard model. A lower number is better because it corresponds with the participant's model more closely aligning with the gold standard model.

The number of transitions was not calculated because in this experimental setup it is nearly always equal to one less than the number of states (except if a model had multiple behaviors with only one state. This edge condition is ignored for this dissertation). Thus, there is a consistent relationship between the number of states and the number of transitions.

Results: Table 6.5 presents the results for this measure, which consist of mean absolute value differences of different model elements between the participants' models and the relevant gold standard model.

Table 6.5: Operationalization 1, Measure 3 Results. Mean absolute value difference between the number of a modeling element in participants' models compared to relevant gold standard model. (Lower is better)

Study Condition	**Explanations**	**Conditions**	**Formulas**	**States**
Control (n = 5)	7.2	35.6	7	7.2
Experimental (n = 3)	11.33	18.33	8.67	7

6.4.2 Operationalization 2

Experimental condition models will be more precise than Control condition models.

Operationalization 2 evaluates model from the perspective of evaluating explanations. Rugaber's [125] *Explanatarium* website lists a set of generic criteria with which to evaluate explanations, with precision being one of them. Rugaber's definitions of precision act as a jumping off point towards constructing a similar, simple definition of precision for SBF* models. Precision in the context of SBF* models is defined here as the complexity or size of the model, for every detail provided by the modeler represents a precise commitment made by the explanation.

This operationalization does not meaningfully relate to notions of internal and external consistency.

6.4.2.1 Operationalization 2- Measure 1

Experimental condition models will have more model elements of a given type than Control condition models.

Consider each model element as a commitment made by the modeler about the modeling topic. Better models thus are those with higher numbers of any given model element type (e.g., explanations), building on the parent operationalization's argument that more commitments equal higher precision. The model elements targeted here specifically relate to the behaviors in the models. The participants' structure submodel and functions were premade and unchangeable, so they provide no discriminatory insight.

This operationalization is very similar to Operationalization 1, Measure 3, and the numbers from that measure were used here. However, this operationalization and its measures instead look at the raw numbers of the model elements, defining a higher number as better. Note that the number of transitions is not considered in this operationalization for the same reason that it was not considered in Operationalization 1, Measure 3.

Results: Table 6.6 presents the results for this measure, which consist of the mean number of modeling elements.

Table 6.6: Operationalization 2, Measure 1 Results. Mean number of model elements. (Higher is better)

Study Condition	Explanations	Conditions	Formulas	States
Control (n = 5)	10	38.2	5.8	12.6
Experimental (n = 3)	4	21.33	3.33	10.67

6.4.3 Operationalization 3

Experimental condition models will be more syntactically correct than Control condition models.

SBF* models have a precise syntax. Having correct syntax is analogous to having correct grammar in a natural language. Thus, more syntactically correct models are better than those with less correct syntax. This operationalization covers this perspective.

This operationalization does not meaningfully relate to either internal or external consistency. Although syntax is important for the model itself to make sense, internal consistency is about how aspects of the model relate to each other rather than the physical construction of the model.

6.4.3.1 Operationalization 3 - Measure 1

Experimental condition models will have fewer formulas with incorrect syntax than Control condition models.

The modeling interface enforced well formed syntax for most situations. However, the formula field in the modeling interface allowed for anything to be written within it. This measure checks the syntax of the formulas.

This measure reports on the number of formulas with syntax errors. The fewer formulas with syntax errors, the better. To arrive at this number, the final version of Simulation in its Complete configuration (see Chapter 5) was ran against each participant model within the modeling interface. Simulation's implementation should fail if any formulas have incorrect syntax or semantics because it will be unable to solve the equation expression. This feedback loop was used to check the formulas. If Simulation threw errors, formulas were manually inspected for syntax and semantic errors.

This measure was skipped for participants whose model had at least one function-subfunction infinite loop. A function-subfunction infinite loop prevented the model from running to completion with Simulation without making ablations to the model.

Results: The mean number of formulas with syntax errors in participant models was 0 for the Experimental condition (n = 3) and 1.25 for the Control condition (n = 4). P8 (Control and Technological) was skipped due having at least one infinite loop, so the n value for Control is reduced by 1.

6.4.4 Operationalization 4

Experimental condition models will be more semantically consistent than Control condition models with regards to their behaviors and functions.

The SBF* modeling language also has semantics associated with it, which refers to the meaning expressed by the language. Better models will be those models that are more consistent with these semantics. Again, only the behaviors get investigated here because the structure and functions were identical and unchangeable and thus do not provide any discriminatory insights.

Connecting to internal and external consistency, this operationalization relates to internal consistency. In checking model semantics, this operationalization can be viewed as (essentially) looking at ways in which parts of the model make sense relative to other parts of the model. The connection with Measure 2 is more tenuous, but one can consider that a function-subfunction infinite loop fails internal consistency because cycles in the functional decomposition (caused by how functions relate to each other in the model) do not make sense semantically.

6.4.4.1 Operationalization 4 - Measure 1

Experimental condition models will have a higher percentage of attribute values of the same type as the attribute's type (i.e., Quantitative and Descriptive[2]) compared to Control condition models.

Each attribute in the experiment was defined as being of the Quantitative or Descriptive type in the structure submodel. Quantitative attributes are numerical, and Descriptive

[2]*Qualitative attributes, while supported by DESC, were not in the Usefulness Study for simplicity.*

attributes are not. Attribute values given in conditions in the behaviors (and in the functions, for that matter) should correspond to these types. This measure inspects to what extent the values written in the model match with their defined types. Superior models have more matching values.

This was investigated by manually looking at condition values in the behaviors and determining if they matched with their attribute's type. For simplicity, Descriptive attribute values were allowed to be anything, so this boiled down to ensuring that all non-numerical values were only assigned to Descriptive attributes.

Results: The mean percent of values that match their attribute's type was 100% for the Experimental condition (n = 3) and 90.83% for the Control condition (n = 5).

6.4.4.2 Operationalization 4 - Measure 2

Fewer Experimental condition models will have a function-subfunction infinite loop through function explanations than Control condition models.

An explanation can point to a function, meaning that function (a subfunction in this context) helps the system being modeled to transition from one state to the next. Given these semantics, the function-subfunction graph formed by these relationships should be a directed, acyclic graph. It makes no sense for what was a superfunction to help the behavior of a subfunction (or one even further down the graph) make a transition. Further, cycles create situations where Simulation will loop endlessly (or, if implemented, until the program breaks).

If a modeler ever composes a model with such a cycle, this is thus a semantic error termed here as a function-subfunction infinite loop. Each participant's model was manually analyzed to determine if it contained at least one infinite loop. Superior models do not have any infinite loops.

Results: The number of participants with at least one function-subfunction infinite loop was 0 for the Experimental condition (n = 3) and 1 for the Control condition (n = 5).

6.4.4.3 Operationalization 4 - Measure 3

Fewer Experimental condition models will have at least one semantic error (erroneous attribute value references or descriptive values in Quantitative formulas) in their formulas compared to Control condition models.

Operationalization 3, Measure 1 investigated the syntax of the formulas within participant models. This measure focuses on semantic issues. Formulas have rich semantics behind them (see Chapter 4). If a formula violates any of these semantics, it counts as having a semantic error.

This analysis was conducted alongside that for Operationalization 3, Measure 1. Please see the description in section 6.4.3.1. If a formula was identified with a semantics error, this counted as the model having at least one formula with a semantics issue. Superior models have no semantic errors in their formulas. As in Operationalization 3, Measure 1, this measure was skipped if the model had any function-subfunction infinite loops.

Results: The number of participants with erroneous formula semantics for at least one formula was 0 for the Experimental condition (n = 3) and 1 for the Control condition (n = 4). P8 (Control and Technological) was skipped due to having at least one function-subfunction infinite loop.

6.4.4.4 Operationalization 4 - Measure 4

Experimental condition models will have a higher percentage of attribute values in behaviors that Simulation agrees with compared to Control condition models.

Simulation verifies the extent to which the conditions in states are consistent with and justified by the explanations on transitions. This is a form of semantic analysis, and thus it is repeated here as a measure in the semantics operationalization. To investigate this measure, the final version of Simulation in its Complete configuration (see Chapter 5) was ran against the participant's model. The number of condition-related issues it found were manually counted, and the number of conditions that the simulator agreed with was determined by subtracting this value from the total number of conditions in the behavior. Superior models will have a higher percentage of agreed-with conditions.

Before doing so, any previously identified syntactic errors within formulas that had obvious fixes not requiring any meaningful interpretation (such as an underscore accidentally in place of a period) were fixed.

This measure was skipped if previous analysis identified any function-subfunction infinite loops. It was also skipped if formulas contained any semantic errors or non-obvious syntactic errors (none of these were found). P3 (Control and Biological) was skipped because they had erroneous formula semantics. P8 (Control and Technological) was skipped because they had an infinite loop.

Results: The mean percent of conditions in the behaviors of a participant's model that Simulation agreed with was 82.86% for the Experimental condition (n = 3) and 86.89% for the Control condition (n = 3).

6.4.4.5 Operationalization 4 - Measure 5

Experimental condition models will have a higher percentage of attribute values in the provides conditions of the model's functions that Simulation can determine are consistent with the results of its behavioral simulations compared to the Control condition models.

This measure follows the same logic as Measure 4. Simulation also verifies the semantics of the model by checking for consistency between the outcomes of the behaviors (i.e., the final attribute values) and the provides conditions of the functions that they achieve.

To be consistent with a provides condition in the function, a final attribute value must (a) be defined and (b) not contradict the function's condition. Since all conditions in the functions are equality statements, part (b) works out to mean that the final attribute values from the behaviors must match the values defined in the provides conditions of the function. If (a) isn't true, Simulation cannot determine what the appropriate value in the function should be. If (b) isn't true, Simulation determines that the value in the function is inconsistent with what is given in the behavior. Superior models will have a higher percentage of provides conditions that Simulation could determine and found to be consistent.

This measure was investigated simultaneously and with the same results as those for Measure 4, except here the number of issues related to functions were counted. Note that attributes that were in the behavior but not in the function were ignored by Simulation.

Results: The mean percent of function provides conditions that Simulation could determine and found consistent was 55.56% for the Experimental condition (n = 3) and 14.29% for the Control condition (n = 3). Like in Measure 4, P3 (Control and Biological) and P8 (Control and Technological) were skipped. P3 had erroneous formula semantics. P8 had an infinite loop.

6.5 Model Conclusions

The previous section presented the results of analyzing the models that participants constructed. This section discusses these results and then touches on the participant who explicitly disregarded some of DESC's output.

Table 6.7 shows the overall results by operationalization and measure, categorizing at a high level which condition (if either) produced superior results. Overall, Experimental condition models were superior to Control condition models. Experimental had superior results in 7 out of 10 operationalization and measure combinations, whereas Control was superior in only 2 and the results were inconclusive in 1. At the operationalization level, Experimental condition models had superior results in 3 out of 4 operationalizations compared to 1 for Control.

These results follow the expected outcome, suggesting that the addition of DESC's evaluative feedback helps people develop better models. To put it another way, these results provide evidence (albeit high-level because the cycle itself was not observed/analyzed) that DESC does indeed support the hypothesized incremental design revision cycle as it

Table 6.7: Overall results of model analysis

Operationalization	Measure	Outcome
1	1	Experimental was better
	2	Experimental was better
	3	Inconclusive. Half in favor of Control, half for Experimental
2	1	Control was better
3	1	Experimental was better
4	1	Experimental was better
	2	Experimental was better
	3	Experimental was better
	4	Control was better
	5	Experimental was better

relates to the external model, providing feedback that leads to revising that model. It makes intuitive sense that having a support tool that helps you reflect on and thereby improve your model would enhance your work product.

However, the Experimental condition was not superior in every measure, and this needs to be accounted for. Specifically, the Control condition produced superior results in Operationalization 2, Measure 1 (precision as defined by counting the number of various modeling elements) and Operationalization 4, Measure 3 (semantic consistency in behaviors as defined by the percent of state conditions that Simulation agreed with).

One hypothesis for Operationalization 2, Measure 1's (precision) outcome: it could be that a trade off between model development speed and quality was created by DESC. In other words, the results may be due to slower modeling velocity. If one periodically pauses to reflect on one's model (via DESC's feedback), this would naturally slow down modeling progress. However, the broader trend suggests that DESC-influenced (i.e., Experimental condition) models are otherwise generally higher-quality. Thus, the trade off suggested here is model size versus quality. Note that this explanation assumes Control condition participants did not spend as much time reflecting on their models, which needs to be confirmed before accepting this hypothesis.

The Operationalization 4, Measure 4 (semantic consistency in behaviors) results are more troubling because Experimental condition participants had access to Simulation that (even though it was an earlier version) gave them feedback about these semantic issues. Why then would Control condition participants do better on this measure?

First, note that two Control condition participants, P3 (Control and Biological) and P8 (Control and Technological), were excluded from Measure 4 (and 5) because of erroneous formula semantics and a function-subfunction infinite loop, respectively. Also, P7 (Experimental and Technological, recategorized to Control and Technological) and P9 (Control and Biological) had formulas with syntax errors that were fixed before generating results for these measures. Thus, in the broader context of the experiment, four out of five Control condition participants had issues related to simulation, whereas none of the Experimental condition participants did. Because the participants with issues were either excluded or had their issues fixed before computing the results, the results for the Control condition in Measures 4 and 5 should be taken with a grain of salt.

That said, one hypothesis to explain these results is that DESC's output was (at least partially) ineffective at triggering participants to correctly repair their models when they were shown to be erroneous. Perhaps the output was confusing to participants and in so doing, caused them to attempt repairs that worsened the situation. A less extreme take on this is to consider that the difference between Control and Experimental conditions on this measure is rather small (a little more than 4 percentage points between them), so perhaps it is simply that the participants could not determine how to solve their semantics issues due to ineffective Simulation output and thus have about the same amount of errors as Control condition participants. In any case, the solution to this situation could be to improve DESC's output (including internally improving how it diagnoses modeling errors) to make it more useful.

Finally, consider the inconclusive results for Operationalization 1, Measure 3 (model similarity versus a gold standard model defined as complexity similarity). In this case, half of the model element types were more similar for the Experimental condition and half for the Control condition. It is difficult to draw an explanation for split results such as these, so they are pointed out here simply to say that the lack of a firm result here suggests need for further investigation with more participants to determine if a meaningful pattern exists.

6.5.1 Disregarding DESC

Participants in the Experimental condition of the Usefulness Study were given access to the automated verification technology. They were asked to document times when they chose to disregard or ignore the technology's advice and why. Only participant P2 chose to produce such documentation. P2's documentation contained a mix of commentary and what appear to be Simulation results.

Their document was analyzed qualitatively to derive themes. The first theme was a belief that the technology was suggesting the wrong answer. The second theme related to the technology being too literal or, to put it another way, unreasonably precise. Finally, the last comment seemed almost like a summary comment and related to both of these themes.

Assuming that P2 is responding to the Simulation results in particular, both of these themes suggest that the participant was not receiving helpful feedback. Unless the participant was seeing bugs in the software, the simulation's suggested values are derived from how it reasons about the model. They are neither arbitrary nor unreasonably precise (since they reflect the participant's choices), nor are they wrong (since they are generated through reasoning about the simulation and not some external source). However, that the participant felt these things implies that Simulation needs to provide better, more explanatory feedback. This would at least contextualize the answers so that they do not appear erroneous or misguided.

P2's feedback is valuable and illuminating. Even this single participant's feedback guides future work to improve the usefulness of DESC. Indeed, upon reflection, improving Simulation's explanations intuitively sounds like a fruitful avenue for future work, and this feedback motivates that work.

6.6 Surveys

This section presents the results of the surveys. They are named here using the same nomenclature as used in Appendix C so readers may easily cross-reference with that appendix to see the contents of the surveys in question.

6.6.1 Initial Survey

The Initial Survey collected demographic information from participants with the intent of capturing details from the participants that provides insight into the people who participated in this experiment.

Zooming out from the specific answers and even away from the answers per condition, the Initial Survey shows that the participants overall were not homogeneous. Questions 2 and 3 reveal a mix of male and female participants, different college degree levels (ranging from Bachelor's to Ph.D.), and both Electrical Engineering and Computer Science majors (ignoring those who did not answer the major question). Similarly, there is some variety in the answers of all parts of question 4 (the number of classes taken in a few fields) although some parts lean heavily towards a particular answer choice. Again, there is overall a variety of answers on question 5 (years of experience in conducting design).

Although variety was not present across all questions for all conditions in the Usefulness Study, overall there is variety in the participants across the whole set. The non-homogeneity of the participant set supports that the findings in this experiment could generalize beyond a single participant profile.

6.6.2 Pre- and Post-Modeling Surveys

The Pre- and Post-Modeling Surveys were identical with each other within each of the Biological and Technological conditions and were designed to be analyzed as a pair. They reflect a kind of pre- and post-test given to participants. The surveys capture a participant's knowledge through two questions. If the hypothesized incremental design revision cycle worked correctly, a participant's external model *and* internal model (cognitive understanding) of the topic should be improved moreso than if this cycle was not supported by the AI agent.

The topic-related parts of the questions differ depending on whether the participant was in the Biological or Technological condition, but the idea behind the questions remains the same. Given this, the surveys will be discussed as if they were the same despite being topic-dependent.

The Pre-Modeling Survey captured the amount of knowledge of the modeling topic that a participant had before the modeling exercise. This sets a baseline for their knowledge. The Post-Modeling Survey captured the amount of knowledge of the modeling topic that a participant had after the modeling exercise. Thus, the difference in results between the Pre- and Post-Survey reflect how the participant's knowledge on the topic being modeled changed due to the modeling exercise.

6.6.2.1 Question 1

The first question asks the participant to rate how well they understand the topic on a scale from 1 to 7, with 1 meaning "I do not understand it at all" and 7 being "I understand it completely." Answers to this question provide a self-reported, discrete understanding score. Such a score is valuable because it is easy to analyze, but the score is also prone to bias and thus cannot be the sole data point from which to draw conclusions.

To analyze this question, the difference (Post-Survey minus Pre-Survey) was calculated for each participant. This difference reflects the direction and scale of change in understanding (if any) that a participant underwent through the modeling exercise. The means of the conditions were then compared.

The mean Post - Pre-test difference in participant responses for question 1 were 1.2 for the Control (n = 5) condition and 2 for the Experimental (n = 3) condition.

6.6.2.2 Question 2

The second question asks the participants to describe the modeling topic. To avoid participants being forced to write nonsense if they did not know anything, a participant was allowed to skip this question on a test if they answered 1 in Question 1 for that test.

Answers to this question were analyzed qualitatively, first using P1's answers as a test and then analyzing the rest of the participants. The technique and codes were refined during the first run of the analysis. Analysis was done without being blind to the participant's condition, but the condition was not taken into consideration. During the first run, participant data was analyzed in order of participant ID (P2 then P3, etc.). To avoid ordering bias, the second run had a randomized ordering.

Coding looked at three dimensions and did so in the following order: (1) size, (2) substance, and (3) correctness. This multidimensional approach allows for nuanced results. Before conducting any of the analysis dimensions, answers were separated into an ordered set of sentences. If the answer was empty, no analysis was done.

The *size* dimension assumes that a larger answer reflects a greater understanding of the topic. Each sentence was coded as either relevant or irrelevant. Size of a participant's answer was measured by counting the number of relevant sentences that it contained. Counting only relevant sentences avoids overcounting due to sentences that did not directly explain the modeling topic. A relevant sentence is about the modeling topic, and an irrelevant sentence is not. Only flagrantly irrelevant sentences were coded as such. P9's (Control and Biological) post-test had the only sentence coded as irrelevant, "A more efficient flat plate will enable trailing vortices that can propel a flat plat towards its intended direction." This sentence was about vortices propelling a flat plat and did not seem related to friction drag.

The mean Post - Pre-test differences for the size dimension were 0.6 for the Control condition (n = 5) and 1 for the Experimental condition (n = 3).

The following substance and correctness dimensions assume that at least one sentence was coded as relevant. If there were no relevant sentences in the answer, substance and correctness analysis was not done.

Substance looks at to what extent the participant's answer refers to behaviors, functions, or structures. Each relevant sentence, as determined by the size dimension, was coded as either structure, behavior-function, or other based on what it is primarily about. When necessary, other sentences in the answer were considered to add context to a current sentence's meaning and intent. A structure-coded sentence is one primarily concerned with details about the components of the topic, connections between those components, and attributes (but not value change) of the components. For example, this sentence from P2's pre-test was coded as structure: "An electric toothbrush consists of a power source, a motor, a frame, a driveshaft, and a brush." A behavior-function-coded sentence is primarily concerned with processes that the topic undergoes and/or objectives, goals, or purposes of the topic or some part within it. For example, this sentence from P3's pre-test was coded as behavior-function: "The friction is generated by the material of the skin (or plate)." If a sentence had both strong structural aspects and clear behavior-function aspects, behavior-function was the favored code. If a sentence did not belong to structure or behavior-function, it was decided whether (a) the sentence was still relevant and should be coded as other or (b) whether I should recode the sentence as irrelevant in the size dimension. A sentence coded as other means that it is relevant but does not fall into the structure of behavior-function categories. For example, this sentence from P3's post-test was coded as other: "The friction drag can be somewhat explained by an analogy of throwing a deck of cards onto a table."

Table 6.8 reports the mean differences (Post-test minus Pre-test) on the number of structure, behavior-function, and other sentences, respectively.

Correctness is the final dimension, and it is essential because it measures the extent to which the participant has an accurate understanding. To test for correctness, each relevant sentence (as determined by the size dimension) was checked against the condition-appropriate gold standard model. When necessary, other sentences in the answer were considered to add context to a current sentence's meaning and intent. Each sentence was

Table 6.8: Mean Post - Pre difference in number of structure, behavior-function, and other sentences for Modeling Surveys' Q2

Condition	Structure Sentences	Behavior-Function Sentences	Other Sentences
Control (n = 5)	0	0.4	0.2
Experimental (n = 3)	-0.33	1.33	0

coded as either (a) completely correct, (b) partially correct, (c) completely incorrect, or (d) unknown. Completely correct means that all the statements in the sentence corresponded to contents of the model and everything in that sentence is confirmed by the contents of the gold standard model. Completely incorrect means the same thing about correspondence but that everything was completely contradicted, or it means that part of the sentence was incorrect but no correspondence could be found for the rest. Partially incorrect means part of the sentence corresponded and was confirmed but the rest either corresponded and was contradicted or could not be confirmed by the model due to lack of correspondence. Correctness and correspondence was approached liberally, looking for general, abstract (especially in the case of values) matches rather than precise equality.

After doing this coding, a normalized score was derived for the participant's answer. Sentences coded as completely correct were given 1 point, partially correct were 0.5 points, and completely incorrect were 0 points. Unknown sentences were not considered in the score. Values were summed and divided by the number of relevant sentences minus the number of unknown sentences, which normalizes the score. Each participant thus earns a score from 0 (everything was completely incorrect) to 1 (everything was completely correct). Normalized scores were rounded at the second decimal place. A participant might have a score whose denominator was zero because all the sentences were coded as unknown. In these cases, the reported score is N/A. Such situations reflect an answer that was not at all covered by the gold standard model. Although not necessarily implying an erroneous answer, this raises questions about the information in the answer because the gold standard model is a relevant model of the modeling topic.

The mean Post - Pre-test difference in participant responses were 0.35 for the Control condition (n = 5) and 0.13 for the Experimental condition (n = 2). The Experimental condition's n-value is reduced by 1 due to P6 (Experimental and Biological)'s removal from having a denominator of zero for their Pre-test normalized score.

6.6.2.3 Analysis

Most of the results for the Pre- and Post-Modeling Surveys produced similar and thus inconclusive outcomes for both conditions. Question 1 results show that the mean differences that the lift provided by the Experimental condition is slightly superior to that provided by the Control condition. This could be due to the feedback provided by DESC enhancing the learning process by supporting the hypothesized incremental design revision cycle.

Regarding Question 2, the size dimension results show us that the Experimental condition answers grew fractionally more than the Control condition ones. In the substance dimension, the differences in changes between the conditions were very small. Note that the substance dimension does not inherently have a better-or-worse aspect to it, but patterns of results may have been informative. The differences in these dimensions were too small to draw any firm conclusions, suggesting at most that the presence of DESC's evaluation techniques does not dramatically skew the abstract structure of the model (i.e., whether one favors more functions or behaviors, etc.).

Finally, let us consider the correctness dimension to Question 2. Based on the mean differences, we see that, surprisingly, the Control condition participants grew more than the Experimental condition participants. Considering that results were normalized to be out of 1, the differences between conditions appears sizable. This suggests that access to the DESC implementation does not enhance learning more so than lacking that access and may in fact inhibit it. This might be the case due to the cognitive overhead of learning a new technology (the techniques of DESC) in addition to learning about the modeling topic. Perhaps if participants were allowed to gain proficiency with the DESC implementation first, it would help their design concept learning. Alternatively, perhaps participants used DESC's feedback to improve their models but did not integrate the changes into their cognitive understanding, suggesting a need to *teach* the right answers rather than *tell* them.

6.6.3 Exit Survey

The participants in the Control condition received a one-question exit survey. The Experimental condition participants received a survey that included that one question (with one word accidentally different) plus several additional more. These extra questions were related to the automated verification technologies that they used and so were not relevant to Control condition participants.

6.6.3.1 Question 1

This question was given to both conditions although the Experimental and Control condition versions accidentally differed by one word. It asked participants to describe at least three ways that they would improve the model building technology that they used. This question serves to drive further refinement of the interface for future research and deployment. Qualitative analysis was conducted on the answers to derive categories, which were as follows: Auto-Populate, Better Equation Editor, Better Training, Better User Feedback For Buttons, More Info Available On Screen At a Time, More Precision, No Issues, Online Help, Visual Design of UI, and Visual Representation (With or Without Interaction).

6.6.3.2 Question 2

This question and all future ones were only given to the Experimental condition. Question 2 asked participants to rate on a scale from -3 (Very Negatively Impacted) to 3 (Very Positively Impacted) to what extent the automated verification technology impacted their model building.

Three participants answered the question, giving scores of 1, 1, and 2.

6.6.3.3 Question 3

This question asked participants to list at least three ways that the automated verification technology (sic) impacted their model building. They were allowed to give both positive and negative answers, regardless of their Question 2 answer. Participants were told to skip this question if they answered 0 (No Impact) to the prior question. Qualitative analysis was again used to derive categories. Although P7 was not normally considered an Experimental condition participant, their results are included in this analysis in case they shed useful insights. P7's results (two instances of Modeling UI Point (Irrelevant to Auto. Verification Tech.)) were relevant to the broader modeling interface, and thus they remain in the findings as legitimate responses.

The categories derived were as follows: Easier Than Verifying With Scientific Paper; Modeling UI Point (Irrelevant to Auto. Verification Tech.); Reference Model Was Confusing; Structured the Thought Process; Useful Initially, But Ignored For Long Term; and Was Not Useful.

6.6.3.4 Question 4

This question asks to what extent did the automated verification technology impact a participant's understanding. Participants were to give a numerical rating on the same scale as in Question 3.

Three participants answered the question, giving scores of 1, 2, and 2.

6.6.3.5 Question 5

In this final question, participants were asked to describe at least three ways that they would improve the automated verification technology. Qualitative analysis was again used to derive categories. Like with Question 3, although P7 was not normally considered an Experimental condition participant, their results are included in this analysis. P7's result (the instance of Auto-Populate) was relevant to the broader modeling interface, and thus it remains in the findings as a legitimate response.

The categories derived were as follows: Auto-Populate; Avoid Spelling Glitches; Continuous Feedback; No Changes; Simplify Interaction With Reference Model; Unsure Why Model Was Incompatible; Vague Error Messages; and Visual Representation of Model. All categories had only a single instance associated with them because no patterns were seen in the data.

6.6.3.6 Analysis

Questions 1 and 5 solicited feedback from participants. The results of these questions provide guidance for how to improve the modeling interface and DESC implementation for future work.

For Question 2, the three Experimental condition participants all gave a score greater than zero (two 1's and one 2), suggesting that they felt the automated verification technology had a positive impact on their model building to some degree. This is a positive finding, suggests that participants generally had a positive (although not strongly so) experience with the DESC implementation, and encourages further research in this area.

Question 3 asked participants to explain how the automated verification technology impacted their model building if they felt it had any impact. The analysis suggested that the participants may have conflated the automated verification technology with the broader modeling interface. This is troubling and calls into question the results of Questions 2 through 5 on this survey. Additionally, the results suggested room for improvement, with a third of the categories being negative. That said, there were also two positive categories across two instances.

Question 4 asked about the automated verification technology's impact on the participants' understanding of their modeling topics, using the same scale as in Question 2. Again, the three Experimental condition participants all gave a score greater than zero (one 1 and two 2's), suggesting a positive impact. This result is interesting considering the Pre- and Post-Modeling Survey results that showed understanding was not necessarily improved with access to the DESC implementation. Perhaps participants *perceived* that DESC was useful in this area even though it was not necessarily, which reinforces that participants had a positive experience with the technology.

6.6.4 Overall Conclusions on the Surveys

Overall, the surveys produced largely inconclusive results relative to Hypothesis 1. Although Experimental condition participants self-reported that they found the DESC implementation had a positive impact on model building and understanding (granted they may have conflated DESC with the modeling interface) and there is a fractionally larger growth for Experimental condition participants versus Control condition participants in self-reported understanding on the Pre- and Post-Modeling Surveys, correctness analysis of the written answers on those surveys showed that Control condition participants grew more in their ability to accurately describe the modeling topic.

This suggests, conservatively, that DESC did not achieve success at supporting the hypothesized incremental design revision cycle because participants did not transfer improved external models into improved internal models (cognitive understanding). It is unclear why the DESC implementation did not transfer to greater correctness given the generally positive model results above and the positive (if only slightly so) self-reported understanding results. Perhaps participants did not have the inclination to integrate feedback from DESC into their internal cognitive understanding of the topic, suggesting a need to consider DESC's human-computer interaction design from a pedagogical perspective. Further investigation is warranted with a larger sample size to determine exactly why this occurs, what can be done about it, and even if the small sample size here reflects a larger pattern or not.

6.7 Overall Conclusions

This final section discusses overall takeaways from the Usefulness Study. Overall, the Usefulness Study findings **partially supported Hypothesis 1**. Participant models in the Experimental (i.e., DESC equipped) condition showed a trend of being superior to Control condition models across the operationalizations and measures, and Experiment condition participants had a somewhat larger self-reported growth in modeling topic understanding compared to Control condition participants on the surveys. These results do not directly comment on the incremental design revision process because they only view the end of the modeling session (relative to the beginning, in the case of understanding). However, they do suggest that the presence of DESC supports the incremental design revision cycle in terms of development of better external models. Additionally, it may increase one's feeling of understanding about the modeling topic.

However, written explanations about the modeling topics were mixed and, importantly, the correctness dimension strongly favored Control condition participants. Additionally, Control condition models were superior on a couple measures. These results suggest that improved external models from the Experimental condition did not necessarily transfer to improved understanding and that the external models were not universally improved from all perspectives, respectively. It appears that DESC does not support the incremental design revision cycle in terms of producing improved internal models (cognitive understanding).

That Experimental condition participants produced superior models but Control condition participants produced more correct written explanations is an issue that needs further investigation in future work. These results imply that refinements in external model building did not transfer to improved understanding. Perhaps participants were merely reacting to DESC and not integrating the changes to their mental models. In any case, DESC had a partial positive impact (external model building), so it partially supports Hypothesis 1.

Overall, these findings warrant a larger, refined study to determine the extent to which the results of this Usefulness Study represent patterns and why exactly certain results happened.

Hypotheses were given earlier in this chapter for why (for example) Experimental condition models had worse behavior condition semantics than Control condition models, and a refined study and its analysis should be conducted to test these hypotheses (or otherwise determine the reason) and to act on that finding by modifying DESC, the experiment design, etc. Beyond running another experiment, one could also conduct further analysis on the existing data by, for example, analyzing the timestamped model snapshots that were saved from this study to determine if the hypothesis about model velocity is accurate and to get into more detail about DESC's role in incremental design revision.

Finally, three points of reflection on the Usefulness Study:

- A multi-dimensional analysis was used to investigate the models built by participants, and the outcome reveals the utility of such a nuanced approached because different conclusions would have been drawn if only one dimension had been used. For example, if only Operationalization 2 had been used, the Control condition would have been viewed as completely superior, and the reverse would occur if only Operationalization 3 had been used.

- Although they are only lightly covered in this dissertation, asking participants for feedback on the modeling interface and automated verification technologies (the latter of which is synonymous with this study's version of DESC) gathered useful hints for future work. These questions, even if they are not directly useful to verifying research hypotheses, are straightforward to ask and provide utility.

- The experiment successfully acted as a proof of concept for how one might situate DESC within a modeling context. A concrete starting point now exists for what such an interaction looks like and what kind of training accompanies that interaction, and future work can improve it.

CHAPTER VII

CONCLUSIONS

This chapter summarizes the dissertation's problem and high-level solution, the formal research questions and hypotheses, and the thesis statement. It reflects on the extent to which the hypotheses are supported by the evaluation of the Design Evaluation through Simulation and Comparison (DESC) AI agent that occurred through the Computational Experimentation and the Usefulness Study in Chapters 5 and 6, respectively. The theoretical and technical contributions of the work are presented. Finally, the chapter discusses potential future work directions.

7.1 Summary

7.1.1 Research Problem & High-level Solution

Biologically inspired design (BID) is a design method where practitioners take inspiration from nature to help address their design challenges, and conceptual design is an early phase of design where a function to structure mapping is initially developed, producing a candidate design. Conceptual design in the BID context may engage knowledge of prior designs and biological source cases. Practitioners may operate with less-than-ideal understandings of these topics, perhaps because the interdisciplinarity of the practice means that one may have expertise in one (e.g., the engineering field associated with the prior designs) but not the other (e.g., biology, for the biological source cases). Additionally, intuition suggests that the candidate design is unlikely to come into being fully formed. Revising these three type of knowledge is desirable because issues may have negative consequences.

This dissertation hypothesizes that an AI agent that can evaluate the internal and external consistency of external models of designs (which represent either the practitioner's understanding or vision of the designs) can support a hypothesized cycle of incremental design revision. A practitioner would develop a model that represents (externalizes) their understanding of a design, give it to the AI agent to get feedback, and then decide whether to make revisions or to stop based on that feedback (and external pressures, etc.). If revisions are made, the cycle may begin again.

Such a hypothesis leads to the focus of this dissertation and its essential research problem: how would this AI agent actually conduct internal and external consistency evaluation to support incremental design revision? The answer presented by this dissertation is the Design Evaluation through Simulation and Comparison (DESC) AI agent, which is a computational theory (implemented as an operational AI agent) to evaluate functional design models in the form of Structure-Behavior-Function Star (SBF*) models. Design here is defined as any one of: a prior technological design, a biological source case, or a candidate technological design.

DESC contains two techniques called Simulation and Comparison. The Simulation technique of DESC evaluates the internal consistency (i.e., it evaluates that parts of the model align with each other) of the model's behaviors and functions by simulating the model and using those results to check claims made by it. Simulation had to resolve two core challenges: (1) computer-readable simulation syntax and semantics for SBF* models and (2) the simulation process, including the processing post-simulation that leads to the evaluation results. The Comparison technique evaluates the external consistency of a model. This is the extent to which it accurately represents its topic. Comparison does so by comparing the given model against an alternative one on the same topic, which given the interdisciplinary BID context, could come from a teammate with a different level or kind of expertise than the design practitioner doing the revising. Any differences detected represent potential areas of concern to be further investigated. Comparison had to resolve the challenge of how to computationally detect salient differences without knowing how and to what extent the two models might differ. As part of addressing this challenge, this dissertation hypothesized a novel analogical mapping technique called Compositional

Mapping that uses problem decomposition and constraint satisfaction.

7.1.2 Research Questions & Hypotheses

This dissertation has the following research questions, each with its own hypothesis or hypotheses.

- **Research Question 1.** How might an AI agent support systematic and repeatable incremental design revision?

  Hypothesis 1. It may do so through checking the internal and external consistency of design models.

- **Research Question 2.** How might an AI Agent check a design model for internal consistency?

  Hypothesis 2.1. It may do so by simulating the model and using the results to check the process and functional claims made by the model. The Simulation technique of DESC illustrates this hypothesis.

  Hypothesis 2.2. The SBF* functional modeling language provides the syntax and semantics needed for automated simulation.

- **Research Question 3.** How might an AI agent check a design model for external consistency?

  Hypothesis 3. It may do so by analogically comparing a given model against another model of the same design, where differences represent potential issues. The Comparison technique of DESC illustrates this hypothesis.

- **Research Question 4.** Analogical comparison implies mapping. How might an AI agent map two models?

  Hypothesis 4. It may do so by decomposing the mapping task, solving the subtasks, and using solutions from earlier subtasks to constrain and inform solutions to future subtasks. Compositional Mapping illustrates this hypothesis.

7.1.3 Thesis Statement

This dissertation's thesis statement is as follows:

> The conceptual phase of biologically inspired design entails incremental design revisions across multiple knowledge entities. An AI agent may support systematic and repeatable incremental design revision by checking the internal and external consistency of functional models of candidate designs as well as biological source cases and prior designs.

7.2 Reflection on Research Questions, Hypotheses, and Thesis Statement

To what extent did the two studies (Computational Experimentation and Usefulness Study) support the above hypotheses and the thesis statement? To answer this question, each hypothesis will be reflected on and then the thesis statement will be reflected on.

Hypothesis 1. While not perfect, this hypothesis was **supported** through both the Computational Experimentation and the Usefulness Study. The Computational Experimentation demonstrated that an AI agent is capable of checking the internal and external consistency of externalized design models. It showed that through implementations of the Simulation and Comparison techniques of DESC, respectively. While not directly observing the hypothesized incremental design revision cycles that may have occurred during it, the Usefulness Study provided preliminary evidence from the outcome of a modeling session that the presence of such an AI agent can support production of superior external design models. However, the Usefulness Study did not show that this led to better internal models. Thus, more work is needed to more deeply analyze DESC's role in supporting incremental design revision and to improve how the DESC AI agent supports internal (cognitive) model improvement.

Hypothesis 2.1. This hypothesis was **supported** by the Computational Experimentation, which demonstrated that an implementation of the Simulation technique of DESC, acting as an illustration of using simulation to check for internal consistency issues, was capable of successfully addressing a set of model ablations relating to internal consistency issues.

Hypothesis 2.2. This hypothesis was **supported** by the Computational Experimentation. The Simulation technique in the Computational Experimentation operates over the Structure-Behavior-Structure Star (SBF*) representation. That Simulation could operate over this representation to conduct automated simulation supports this hypothesis.

Hypothesis 3. While there was room for improvement, this hypothesis was **supported** by the Computational Experimentation. The experimentation demonstrated that the Comparison technique of DESC was reasonably capable of not identifying invalid differences when comparing two models. This is a baseline capability for external consistency checking by analogical comparison that sets the stage for further refinement of this work, for the goal is to identify only salient differences as feedback for the design practitioner.

Hypothesis 4. This hypothesis was **partially supported** by the Computational Experimentation. Specifically, Compositional Mapping, which illustrates the ideas raised by this hypothesis, was shown to be decent as a mapping technique for the purposes of DESC's Comparison technique. However, implicit in this hypothesis is also that Compositional Mapping is a superior technique compared to one without its ideas. The Computational Experimentation showed that Compositional Mapping was slightly superior for the purposes of Comparison when compared to a more conventional analogical mapping technique illustrated by Uniform Mapping. This is a positive outcome, but more work is needed to fully support this implicit claim.

The results of this dissertation thus provide at least some positive support for all of the hypotheses. More research on improving Compositional Mapping over Uniform Mapping is warranted for Hypothesis 4 since it is only partially supported. Additionally, while DESC does appear to contribute to incremental design revision when it comes to models, it does not appear to cause improved understanding, which is an area that needs improvement. However, the results nevertheless show that DESC is pursuing a promising direction.

The thesis statement is **supported** by this dissertation in two ways. First, the Computational Experimentation illustrated that an AI agent can check the internal consistency (via DESC's Simulation technique) and external consistency (via DESC's Comparison technique) of candidate designs as well as biological source cases and prior designs. The results showed that both techniques were at least decent at these tasks for the class of problems tested in that experimentation. Second, the Usefulness Study provided preliminary evidence that access to such an AI agent may lead to superior external models of topics thematically related to biological and technological designs although this did not appear to translate to superior internal models. Thus, while DESC was not completely successful at supporting incremental design revision, it produced positive results for evaluating design models and promising results at helping people in a modeling context.

7.3 Contributions

This section discusses the theoretical and technical contributions of this dissertation. Theoretical contributions are those that exist at the level of ideas, whereas technical contributions are artifacts (computer programs, languages, etc.).

7.3.1 Theoretical Contributions

The theoretical contributions of this dissertation get to how it conceptually breaks new, valuable ground in related fields.

This dissertation contributes a novel AI agent for design evaluation in the service of incremental design revision in the context of conceptual design in biologically inspired design. The designs evaluated encompass source cases, prior designs, and candidate designs.

This AI agent, Design Evaluation through Simulation and Comparison (DESC), evaluates functional model of designs, specifically in the form of Structure-Behavior-Function Star (SBF*) models. It evaluates a model's internal consistency by simulating the modelâĂŹs behaviors, and it evaluates the model's external consistency by comparing it with an alternative model on the same topic.

Biologically inspired design (BID) may engage knowledge of biological source cases, prior designs, and candidate designs (and possibly others not covered here). Each of these may contain errors that in turn may impact the conceptual design reasoning process or later stages of the design process. In the conceptual design field and the case-based and analogical design fields, DESC is the first work, based on those closely surveyed, that evaluates all three of these knowledge types.

In BID, DESC appears to be only one of three computational (or computationally-informed) tools to address evaluation, thus inherently contributing to a small body of research on this topic. Given that BID is an interdisciplinary technique and practitioners may not have expertise in either the biological or engineering sides of the designing, the possibility exists for misconceptions and incomplete knowledge. This makes DESC well suited to this domain by evaluating the prior designs and biological source cases used in designing, which neither of the existing two tools do. DESC also evaluates the output of the conceptual design phase, which again neither of the existing tools do. That said, the other two tools address the mapping or the applicability of the biological source, capturing a different (and complementary) set of possible errors. DESC is thus different from this prior work and useful for the domain.

In the field of design critics, DESC is the first work among those closely surveyed to explore both the source case and prior design in addition to the candidate design. Also, the design critics surveyed typically used rules for their evaluation, with one also using comparison in a way similar to DESC. DESC is thus unique in its use of both simulation and comparison together. Further, the Simulation technique is itself novel on its own. Thus, DESC expands the scope of design critics into different knowledge entities and does so with a novel use of two techniques, one of which (Simulation) is novel on its own.

From the perspective of using simulation to verify functional models, the surveyed articles typically do so using qualitative reasoning (although there may exist work that does quantitative simulation or a mix of quantitative and qualitative simulation). This makes sense for early stages of understanding where knowledge is still being developed. However, what if one also had quantitative knowledge and/or wanted to upgrade/transition qualitative models to include quantitative information? For example, one is moving from a vague understanding of a design concept to a more precise understanding. Checking such precise, quantitative values can help ensure that these beliefs about the system make sense and are correct (since they too could have issues), whereas without quantitative reasoning the evaluation would be forced to remain at the more abstract, qualitative knowledge level. Relative to the closely surveyed works, DESC uniquely supports both qualitative and quantitative knowledge (via essentially algebraic equations) and reasoning in simulation. This facilitates both the early stages of knowledge development (qualitative) and allows for intermixing or adding/replacing more detailed information (quantitative), all within the same modeling and evaluation framework.

Furthermore, existing work surveyed in this area tends to generate results for the complete model (or for the entirety of what part was simulated), which has the potential to grow difficult digest as models increase in size. Although DESC's internal simulation reasoning may incorporate multiple parts of the model (e.g., in processing subbehaviors), it organizes and presents the results on a per-function and per-behavior basis. This should help practitioners focus their attention on individual behaviors and functions (units of the greater model) when cognitively processing evaluation results. This process will be aided if they address issues from the lowest level behaviors and upward so that subbehaviors are not unknowingly producing erroneous results.

In addition to a theory of evaluation, this dissertation presents a version of the Structure-Behavior-Function (SBF) modeling language that was co-developed by the dissertation author and is called **Structure-Behavior-Function Star (SBF*)**. SBF* adds syntax and semantics to SBF (and via the reasoning algorithms enacted by DESC) to facilitate the automated simulation done by the Simulation technique of DESC. The experimentation and model-building done in this dissertation illustrates the languageâĂŹs usability and the kind of knowledge it is capable of representing.

This dissertation also contributes to computational analogical mapping–that is, how to computationally align two concepts. Compositional Mapping is novel compared to the closely surveyed techniques by decomposing the mapping problem into a series of subproblems to be solved in a coordinated (through constraints) fashion and where subproblem solvers can be tailored to the associated representation partition, whereas two of the surveyed prior works approach the mapping problem all-at-once, albeit in different ways, and the third surveyed work (while similar at one level) can be thought of as two, related episodes of analogical reasoning. Additionally, in developing the Uniform Mapping approach that uses the Structure-Mapping Engine (SME) [44] to contrast against Compositional Mapping, this dissertation tacitly explores the utility of SME (via Case Mapper's [119] implementation) at mapping two SBF* models to facilitate DESC's Comparison technique and found encouraging results.

Overall, DESC is unique amongst all closely surveyed works in its total combination of *what* it evaluates and *how* it evaluates them. No surveyed prior work has the particular combination of evaluating the biological source case, prior designs, and candidate design using both simulation and comparison. In so doing, DESC contributes a novel approach to computational evaluation in design.

7.3.2 Technical Contributions

The technical contributions of this dissertation reflect produced artifacts. The contributions can be mostly divided along two categories: DESC and SBF*.

Regarding the Design Evaluation through Simulation and Comparison (DESC) AI agent, this dissertation contributes an operational implementation of the AI agent, written in the Java programming language. The high-level technical architecture of this agent is described in Chapter 4. The Simulation and Comparison techniques (and supporting code) are capable of processing Structure-Behavior-Function Star (SBF*) models and producing human-readable output. Included in this contribution are implementations of both Compositional Mapping and Uniform Mapping. Where interfacing with Case Mapper [119] is needed, an automated interface was developed on DESC's side to interact with Case Mapper's server interface. All of this implementation facilitates further investigation by future researchers who can concretely test the current version of DESC with novel input and future versions of DESC with updated implementations derived from this one.

Regarding SBF*, the dissertation author co-developed a formal language definition for this modeling language, which is given in Appendix A. This facilitates reproduction of this work and the use of this knowledge representation in other works. Six illustrative models in this formal language, five of which with topics derived from real-world examples, were also developed or co-developed in the case of Electric Toothbrush and Friction Drag. These may be used by future researchers as examples for their own reasoning strategies. A language definition used to represent the parts of SBF* models relevant to this dissertation in Case Mapper was also developed (see Appendix B). This may aid future exploration into applying SBF* models to the Structure-Mapping Engine [44] mapping technique.

Finally, combining these two categories, an operational web application for constructing SBF* models and getting feedback on them from DESC was developed for the Usefulness Study. This could support future experimentation, and more interestingly, it could be refined and enhanced by future researchers to explore situating DESC and SBF* modeling in design contexts.

7.4 Future Work

This section of the dissertation discusses potential areas of future work for this research.

The most obvious area for future work is to revisit the Usefulness Study. In addition to running the study with more participants to understand what results represent legitimate patterns and what were artifacts of the sample, the Design Evaluation through Simulation and Comparison (DESC) AI agent implementation and the model-building interface could be improved. The results of the current Usefulness Study can inform how one might improve these. A more detailed look into the revising process would also be useful to better inspect the hypothesized incremental design revision cycle. The modeling interface already took snapshots of the model under development; analyzing these snapshots is one way to inspect this. One could also imagine a version of the experiment where the participant was surveyed about their knowledge more than once to capture the notion of incremental revision over time.

The Computational Experimentation and Usefulness Study scoped out the broader biologically inspired design context. Although this provided a useful simplification to focus this dissertation, a fruitful line of future research would be to investigate DESC in a design context. Perhaps one could integrate a DESC implementation in an instance of a design team conducting the conceptual design phase of biologically inspired design, which would explore the (perhaps unexpected) ways in which the technique gets used in a design context and would further test its usefulness and capabilities at supporting incremental design revision.

Along similar lines, another area to investigate DESC deeper would be to test it against models developed without the involvement of the dissertation author. Some of this was done coincidentally through the Usefulness Study, but a deeper exploration with topics selected by participants and complete models constructed by them would be interesting.

This could be viewed in performance terms: how well does DESC work in light of this new, more natural data? It could also be a means to improve the techniques, guiding work into what DESC should be able to detect for classes of model construction issues, issues in internal understanding reflected in the models, and comparison-based issues if multiple people model the same topic, all informed by these models constructed by other people (as a source of more realistic data).

This research is contextualized in biologically inspired design, but model evaluation could apply to other design contexts. For example, to what extent could Simulation generalize beyond concepts related to physical systems (e.g., the design of software)? How would that reasoning process look? What about Comparison? Comparison likely has more potential for generalization because it depends less on the content of the concept representation. That is, Comparison more plausibly could occur outside of functional models. However, Compositional Mapping depends on a representation's ability to be partitioned, so this may place some constraints on its use if one wishes to pair Compositional Mapping with Comparison.

7.4.0.1 Simulation Technique of DESC

Specifically regarding the Simulation technique of DESC, one could explore how to enhance the qualitative equation reasoning aspect of this technique to address ambiguous situations, such as (for example) whether increasing the qualitative value of Medium means it should now become Large or still remain at Medium. In so doing, the simulator could project a space of potential simulation trajectories that could then be compared against the values given in the model. (Note that exploration of this topic already began in the DESC implementation but was conceptually scoped out of this dissertation.) Such a Simulation feature raises even more questions: (a) how does one present the potentially quite complicated results of such an envisionment? (b) At what point does the inputted model have a legitimate issue? Is a value wrong if a single trajectory contradicts it, if a majority of the trajectories contradict it, or some other circumstance? These questions would be interesting to explore as future work for Simulation.

One of the limitations given in Chapter 1 is regarding the inability of the Simulation technique to handle nonlinear behaviors. Consider three types of nonlinear behaviors: (a) multiple independent graphs, (b) branches, and (c) cycles. Multiple independent graphs: a behavior consists of more than one interconnected subgraphs (where a subgraph is a set of states connected together by transitions), perhaps representing multiple ongoing processes that are conceptually related but modeled separately. Branches: a state can have more than one outgoing transition, perhaps representing alternative paths that the behavior could take given different situations. Cycles: a transition can point to an earlier state in the behavior (or the back to the state it extends from), perhaps representing a part of the process that repeats before moving on to a later state (e.g., a train generating speed towards a maximum velocity).

As implemented, the Simulation technique of DESC would not be capable of handling any of the three nonlinear types for various reasons. There is also a theoretical issue raised by branching. If branches are intended to represent alternative paths through the behavior (i.e., a given instance of a behavior traverses a single path rather than all paths), how should function-explanation reasoning interpret the output of a behavior with regards to resolving attribute values during simulation? If a given behavior's output is ambiguous (i.e., one of multiple options) then Simulation perhaps needs to consider ambiguous outcomes. Interestingly, this connects to the above discussion about ambiguity in qualitative equations. Future work that thus considers such theoretical issues with qualitative equations may also wish to consider them with regards to alternative paths through behaviors.

Lest this discussion about nonlinear behaviors be exclusively negative, one can speculate what modifications to make to enable nonlinear behaviors in Simulation. Regarding branching, one could explore ambiguous transition explanation outcomes, projecting an envisionment (a state space) for all possible paths through the behavior. One (computationally challenging) approach to analyzing such an envisionment for evaluation is to compare a given behavior to each generated path through the state space and present results for the path with the fewest issues, which is most likely to be the path that the modeler (the design practitioner) intended. A tie break process would need to be determined if multiple paths have the same number of issues.

If multiple independent graphs represent a single behavior trajectory (i.e., they are not alternatives), Simulation should only need to account for multiple `Start` states in a behavior, something it cannot currently do. This should make it capable of addressing this type of nonlinear behaviors with the non-alternatives assumption. If however they do represent alternative trajectories, a similar approach to what was proposed for branching could be used with each independent graph representing an alternative path (or set of alternative paths if mixed with branching), assuming each independent graph is supposed to be interpreted as such.

Cycles could be achieved by modifying the SBF* language (and Simulation's reasoning to handle the changes) in the following three ways: (1) any state may have multiple outgoing and multiple incoming transitions with the exception of the `Stop` states that may not have any outgoing transitions. (2) Create a new explanation type called `Precondition` that holds an equation of the form `pc: <expression> <operator> <expression>`. For example, a precondition explanation might be `Precondition PC1, described as "pc: Train.Velocity < Train.Velocity.High"`. A Precondition acts as a guard on a transition. Simulation will not traverse a transition unless all of its Precondition explanations evaluate to true. Because of this, all attributes referenced in Preconditions must refer to values in the outgoing, `Before` state. And (3) An attribute value in a condition can now be a comma separated, ordered list of values, with the nth value in the list representing the value for that attribute during the nth time the state gets traversed.

Features (1) and (2) essentially incorporate the programming concepts of branches (if-statements) and loops into behavior graphs. Feature (2) also enhances the concept of branching to only selectively traverse branches. A modeler must be careful, however, to use Preconditions to prevent infinite cycles and to establish Preconditions such that there will always be at least one traversable branch (except for from a `Stop` state) to prevent prematurely ending behavior traversal. Feature (3) accommodates cycles by enabling a modeler to declare all the values (in order) that an attribute will have as traversal crosses it multiple times. One can speculate an addition convenience feature (4) that if there are `m` listed values and the state gets traversed `m+1` or greater times, one should still use the `mth` value for all traversals > `m`. This makes it so that unchanging attribute values (or values that reach some upper bound and stop changing) need not be repeated

7.4.0.2 Comparison Technique of DESC

One area to improve the Comparison technique of DESC is to improve the Compositional Mapping algorithm that it can use to map two models. Regarding Compositional Mapping, one could identify the subalgorithm(s) associated with the invalid differences detected during the Computational Experimentation and work to improve it/them. Future testing with more/different model differences may reveal more subalgorithms that need improvement. Compositional Mapping allows one to optimize specific subalgorithms because of how it decomposes the analogical mapping problem.

7.4.0.3 Considerations for Both Techniques of DESC

Finally, considering both techniques of DESC, it would be interesting to add learning into their reasoning strategies. For example, Compositional Mapping can have interactive subalgorithms. Perhaps the human intervention captured by these subalgorithms could be used to inform future attempts to map with the same models. Alternatively, if a human reasoner could provide feedback about the quality of the results produced by either technique, the techniques could adapt to this feedback. For example, if feedback said that a particular Simulation result was invalid, the technique could omit that result from future outputs.

Another open area for future investigation would be to see if the Simulation and Comparison techniques of DESC could inform each other. Right now, they conceptually complement each other but never literally interact with each other. If one could find some way to leverage the results generated by one technique to support the other, perhaps better results could be found. For example, Simulation could first be ran against the two models to be given as input to Comparison, and the results could be used to increase the amount of information in the behavior states of both models by using implicit value forwarding to add component attributes (values are not considered in mapping) that the modelers left out. This increase in information might help Comparison align the two models.

APPENDIX A

THE STRUCTURE-BEHAVIOR-FUNCTION STAR (SBF*) LANGUAGE DEFINITION

This appendix presents the formal language definition for the Structure-Behavior-Function Star (SBF*) modeling language. All internal comments have been removed from this definition, and it has been formatted for this document. This definition is presented without commentary because it is intended merely to aid in recreating and furthering the work in this dissertation. For a discussion of the SBF* language, see Chapter 3. Historically, the particular syntax here is a revision of the original SBF* language definition that was co-developed with members of the Design & Intelligence Lab at Georgia Tech, including Dr. Spencer Rugaber.

This language definition was used to formulate the models used in this dissertation, and the implementation of the Design Evaluation through Simulation and Comparison (DESC) AI agent ingested and understood the models through this definition. The language is defined in the format used by the Xtext [174] extension to the Eclipse IDE, which automatically generated the Java classes that allowed the implementation to interface with the models.

Note that, while the complete language is defined here, some parts of this total language (such as the environment submodel) were not used in this dissertation.

```
grammar org.dilab.SBFStar with org.eclipse.xtext.common.
    Terminals

generate sbfStar "http://www.dilab.org/SBFStar"
```

```
ModelFile:
 (('Also' | 'also') 'include' importedFiles+=FILEPATH)*
 (sbfModels+=SBFModel) (sbfModels+=SBFModel)*
;

terminal IDENTIFIER:
 ('a'..'z' | 'A'..'Z') ('a'..'z' | 'A'..'Z' | '0'..'9' | '_' |
     '-')*
;

terminal FILEPATH:
 '"' ('a'..'z' | 'A'..'Z' | '0'..'9' | '_' | '-' | '.' | '/' |
     ' ')+ '.sbfstar' '"'
;

Attribute:
 'named'? name=IDENTIFIER
 'of' 'type' type=Type
 ('and' 'unit' unit=Unit)?
 (','? 'described' 'as' ','? description=Description)?
;

Behavior:
 name=Qualified_or_Identifier 'is' 'a' 'behavior'
 (','? 'described' 'as' ','? description=Description)?
 (states+=State | transitions+=Transition)*
;
```

```
BehaviorModel:
 (',''? 'described' 'as' ','? description=Description)?
 (behaviors+=Behavior)+
;

Component:
 name=IDENTIFIER ('is' | 'are') 'a' 'component'
 (','? 'described' 'as' ','? description=Description)?

 ((('It' 'has' | 'They' 'have') 'a' 'function' 'named'?
     functions+=IDENTIFIER)
  | (('It' 'has' | 'They' 'have') 'a' ('sub-model' | 'sub' '
      model') 'named'? subModel=IDENTIFIER)
  | (('It' 'has' | 'They' 'have') ('a' | 'an') 'attribute'
      attributes+=Attribute)
  | (('It' 'has' | 'They' 'have') 'a' ('connecting-point' | '
      connecting' 'point') connectingPoints+=ConnectingPoint)
 )*
;

Condition:
 primitiveCondition=PrimitiveCondition |
 ('(' primitiveCondition=PrimitiveCondition ')') |
  ('not' notCondition=Condition) |
  ('any' 'of' '(' orConditions+=Condition ('or' orConditions+=
      Condition)+ ')') |
  ('(' andConditions+=Condition ('and' andConditions+=Condition
      )+ ')')
```

```
;

Condition_Atom:
  value=Value |
  qualifiedName=Qualified_Name
;

ConnectingPoint:
  'named'? name=IDENTIFIER
  (','? 'described' 'as' ','? description=Description)?
;

Connection:
  firstConnectingPointQualified_Name=Qualified_Name
  ('is' | 'are')? ('a' | 'an')?
  mechanism=Mechanism
  secondConnectingPointQualified_Name=Qualified_Name
  'is' 'a' 'connection' 'named'? name=IDENTIFIER
  (','? 'described' 'as' ','? description=Description)?
;

Description:
  STRING
;

DomainPrinciple:
  name=IDENTIFIER (','? 'described' 'as' ','? description=
      Description)?
```

```
;

EnvironmentComponent:
 name=Qualified_Name ('is' | 'are') 'a' 'component'
 (','? 'described' ('it' | 'them') 'as' ','? description=
     Description)?
 ((('It' 'has' | 'they' 'have') 'a' 'function' 'named'?
     functions+=Qualified_Name) |
  (('It' 'has' | 'they' 'have') 'a' 'attribute' attributes+=
      Attribute) |
  (('It' 'has' | 'they' 'have') 'the' ('sub' 'model' | 'sub-
      model') 'named'? subModel=IDENTIFIER) |
  (('It' 'has' | 'they' 'have') 'a' ('connecting' 'point' | '
      connecting-point') connectingPoints+=ConnectingPoint))*
;

EnvironmentModel:
 ('named' name=Qualified_or_Identifier)?
 (','? 'described' 'as' ','? description=Description)?
 ((components+=EnvironmentComponent)
  | (substances+=EnvironmentSubstance)
  | (stimuli+=Stimulus)
 )+
;

EnvironmentSubstance:
 name=Qualified_Name ('is' | 'are') 'a' 'substance'
 (','? 'described' 'as' ','? description=Description)?
 ((('It' 'has' | 'they' 'have') 'a' 'function' 'named'?
     functions+=Qualified_Name) |
  (('It' 'has' | 'they' 'have') 'a' 'attribute' attributes+=
      Attribute) |
  (('It' 'has' | 'they' 'have') 'the' ('sub' 'model' | 'sub-
      model') 'named'? subModel=IDENTIFIER))*
;

Equation:
 name=IDENTIFIER (','? 'described' 'as' ','? description=
     Description)?
;

Explanation:
 primitiveExplanation=PrimitiveExplanation |
 ('(' primitiveExplanation=PrimitiveExplanation ')') |
  ('not' notExplanation=Explanation) |
  ('any' 'of' '(' orExplanations+=Explanation ('or'
      orExplanations+=Explanation)+ ')') |
  ('(' andExplanations+=Explanation ('and' andExplanations+=
      Explanation)+ ')')
;



Function:
 name=Qualified_or_Identifier 'is' ('a' | 'another') 'function'
 (','? 'described' 'as' ','? description=Description)?
```

```
  (('It' 'is' 'denoted' 'by' 'the' 'verb' 'named'? functionVerb=
      FunctionVerb)? &
   ('It' 'requires' (('the' 'following')? ':'? requiresCondition
       =Condition | requiresNothing ?= 'nothing'))? &
   ('It' 'provides' ('the' 'following')? ':'? providesCondition=
       Condition) &
   ('It' 'is' 'achieved' 'by' 'the' 'behavior' 'named'? behavior
       =IDENTIFIER))
;

FunctionModel:
  (','? 'described' 'as' ','? description=Description)?
  (functions+=Function)+
;

FunctionVerb:
  STRING
;

LogicalExpression:
  STRING
;

Mechanism:
  IDENTIFIER
;

PrimitiveCondition:
  (leftAtom=Condition_Atom relOp=RelOp rightAtom=Condition_Atom)
       |
  ((soloAttributeQualified_Name=Qualified_or_Identifier
   | connection=Connection)
   'is' 'true')
;

PrimitiveExplanation:
  'the'
  (
   (('domain'? 'principle' | 'domain-principle') 'named'?
       domainPrinciple=DomainPrinciple) |
   ('equation' 'named'? equation=Equation) |


   ((
    ('function' 'named'? functionQualified_Name=
       Qualified_or_Identifier) |
    ('state' 'named'? stateQualified_Name=Qualified_Name) |
    ('transition' 'named'? transitionQualified_Name=
       Qualified_or_Identifier) |
    ('connection' 'named'? connectionQualified_Name=
       Qualified_or_Identifier) |
    ('external'? 'stimulus' 'named'? stimulusQualified_Name=
       Qualified_or_Identifier) |
    (('domain'? 'principle' | 'domain-principle') 'named'?
       domainPrincipleQualified_Name=Qualified_Name) |
    ('equation' 'named'? equationQualified_Name=Qualified_Name)
       |
    ('behavior' 'named'? behaviorQualified_Name=Qualified_Name)
        |
    (('data' 'condition' | 'data-condition') dataCondition=
        LogicalExpression)
  ) (','? 'described' 'as' ','? description=Description)?)

 )
;

Qualified_Name:
 IDENTIFIER '.' IDENTIFIER ('.' IDENTIFIER)*
;

Qualified_or_Identifier:
 (Qualified_Name | IDENTIFIER)
;

RelOp:
 ('<' '='?) | ('>' '='?) | ('!'? '=')
;

SBFModel:
 name=Qualified_or_Identifier 'is' ('a' | 'an') 'SBF' 'model'
 (','? 'described' 'as' ','? description=Description)?
 (
  ('The' 'SBF'? 'model' 'has' 'a' 'structure' 'model'?
      structureModel=StructureModel)?
  & ('The' 'SBF'? 'model' 'has' ('a' 'set' 'of')? 'functions'
      functionModel=FunctionModel)?
  & ('The' 'SBF'? 'model' 'has' ('a' 'set' 'of')? 'behaviors'
      behaviorModel=BehaviorModel)?
  & ('The' 'SBF'? 'model' 'has' 'an' 'environment' 'model'
      environmentModel=EnvironmentModel)?
 )
;

State:
 name=IDENTIFIER 'is' ('a' | isStartState ?='the' 'start' |
     isStopState ?='the' 'stop') 'state'
 (','? 'described' 'as' ','? description=Description)?
 ('It' 'has' 'the' 'following'? 'condition' ':'? condition=
     Condition)?
;

Stimulus:
 name=Qualified_Name 'is' ('a' | 'an') 'external'? 'stimulus'
 (','? 'described' 'as' ','? description=Description)?
 ('It' 'has' ('a' | 'an') 'attribute' attributes+=Attribute)*
;

StructureModel:
 ('named' name=IDENTIFIER)?
 (','? 'described' 'as' ','? description=Description)?
 ((components+=Component)
  | (substances+=Substance)
  | (connections+=Connection))+
;

Substance:
 name=IDENTIFIER ('is' | 'are') 'a' 'substance'
 (','? 'described' 'as' ','? description=Description)?

 ((('It' 'has' | 'They' 'have') 'a' 'function' 'named'?
     functions+=IDENTIFIER)
  | (('It' 'has' | 'They' 'have') 'a' ('sub-model' | 'sub' '
      model') 'named'? subModel=IDENTIFIER)
  | (('It' 'has' | 'They' 'have') ('a' | 'an') 'attribute'
      attributes+=Attribute)
 )*
;

Transition:
 ('There' | name=IDENTIFIER) 'is' 'a' 'transition' 'from'
 sourceState=IDENTIFIER
 'to'
 targetState=IDENTIFIER
 (','? 'described' 'as' ','? description=Description)?
 ('It' 'has' 'the' 'following'? 'explanation' ':'? explanation=
     Explanation)?
;

Type:
 IDENTIFIER
```

```
;

Unit:
  IDENTIFIER
;

Value:
  (INT | (INT '.' INT) | STRING)
;
```

APPENDIX B

THE STRUCTURE-BEHAVIOR-FUNCTION STAR (SBF*) LANGUAGE DEFINITION FOR CASE MAPPER

This appendix presents the language definition to represent Structure-Behavior-Function Star (SBF*) models (specifically: the parts that were deemed relevant for this purpose) in Case Mapper's [119] knowledge representation. This definition is presented without further comment. It is provided to help recreate this body of work and also to be transparent about how SBF* models were represented in Case Mapper, for this representation undoubtedly impacted the dissertation results involving it.

Note that the language definition here uses the term "aems" in reference to the old name of the Design Evaluation through Simulation and Comparison (DESC) AI agent in this thesis. This is kept here to be consistent with the actual language definition used in the Computational Experimentation and implementation. If the reader prefers to replace "aems" for "desc" in this language definition in their own research, it should not impact any results.

To utilize this language definition, insert it into a plain text file. Next, via Case Mapper's graphical user interface, open a knowledge base and then import that text file as a "meld" file.

```
(in-microtheory UniversalVocabularyMt)

(isa aems-component Collection)

(isa aems-attribute Relation)
```

```
(arity aems-attribute 4)

(isa aems-connecting-point Relation)
(arity aems-connecting-point 2)

(isa aems-connection Relation)
(arity aems-connection 4)

(isa aems-function Collection)

(isa aems-function-subfunction Relation)
(arity aems-function-subfunction 2)

(isa aems-function-requires-primitive-condition Relation)
(arity aems-function-requires-primitive-condition 6)

(isa aems-function-provides-primitive-condition Relation)
(arity aems-function-provides-primitive-condition 6)

(isa aems-function-verb Relation)
(arity aems-function-verb 2)

(isa aems-function-behavior Relation)
(arity aems-function-behavior 2)

(isa aems-behavior Collection)

(isa aems-state Relation)
```

```
(arity aems-state 2)

(isa aems-transition Relation)
(arity aems-transition 4)

(isa aems-state-primitive-condition Relation)
(arity aems-state-primitive-condition 6)

(isa aems-primitive-explanation Relation)
(arity aems-primitive-explanation 4)

(isa aems-start-state Relation)
(arity aems-start-state 1)

(isa aems-stop-state Relation)
(arity aems-stop-state 1)
```

APPENDIX C

USEFULNESS STUDY MATERIALS

This appendix presents recreations of the materials used for the Usefulness Study (Chapter 6). These are recreations because formatting may not be identical with the actual materials used and superficial details (e.g., a space for the Participant ID; the title of the survey) are omitted for simplicity. However, the meaningful contents are the same.

Regarding the source materials used in the study, the reader will find brief descriptions of them rather than recreations. The source materials came from others' work, so it is more appropriate to provide a description and a reference to their work rather than copying their contents into this document.

The models used in the Usefulness Study can be found in Appendix D under the names Synthetic Car, Friction Drag, and Electric Toothbrush[1]. The Synthetic Car model corresponds to the synthetic model used during training. The Friction Drag model acted as the gold standard model during the Biological condition. Additionally, a version of it with the behavior contents removed and replaced with placeholder information was used as the starting model for participants in that condition. The Electric Toothbrush model acted as the gold standard model during the Technological condition. Similar to the Friction Drag model, a version of it with the same replacement done was used as the starting model for the participants in that condition.

Initial Survey

Below is the set of questions asked in the Initial Survey.

[1]Note that due to organizational issues, the versions of Friction Drag and Electric Toothbrush may accidentally be different than those used in the experiment. However, it is believed that they are the same.

Survey

1. What is your name?
2. What is your gender? Please circle one.

   (a) Male (b) Female (c) Other
3. If you are currently in school, what degree are you in school to earn? Please circle one and provide your name, e.g., Computer Science.

   (a) Bachelor's (b) Master's (c) Ph.D.

   Major:
4. How many classes have you taken in college (both undergraduate and graduate) about the following subjects? For each category, please circle one.

   a. Biology

   (i) Zero classes (ii) 1 to 3 classes (iii) More than 3 classes

   b. Engineering

   (i) Zero classes (ii) 1 to 3 classes (iii) More than 3 classes

   c. Design (except biologically inspired design or biomimicry)

   (i) Zero classes (ii) 1 to 3 classes (iii) More than 3 classes

   d. Biologically inspired design or biomimicry

   (i) Zero classes (ii) 1 to 3 classes (iii) More than 3 classes
5. How many years of experience do you have in conducting design, including biologically inspired design or biomimicry? (Please circle one.)

   (a) Zero years (no experience) (b) Up to 1 year (c) 1-3 years (d) Greater than 3 years

Training Script

Below is the training script that the proctor (which was the dissertation author) used during participant training. Although this script was embellished during training, it formed the basis of the training procedure.

Script

Procedure

1. Consent form
2. Initial Survey
3. Training
4. Modeling Prompt
5. Pre-Survey
6. Function-subfunction relations
7. Build model – up to 45 minutes by may end early
8. Post-Survey
9. Exit survey

[Modeling Training for Usefulness Experiment. Read this document aloud to participants then take questions afterwards. Do not read aloud anything within square brackets ([ ]), such as this.]

[Have this web interface on the screen and load/input the example car model. Refer to it during this.]

In this experiment, you will model part of a biological or technological system based on source material about that system. To model a system means to express your understanding of that system in a systematic, structured way. We want the model you build to reflect your understanding of the system, so you may add detail to the model beyond what is in the source material if you know more.

We build models in modeling languages. I am now going to teach you a subset of one called the Structure-Behavior-Function language or SBF for short. You will create a model in SBF in this experiment. SBF models the structure, behaviors, and functions of a system. I will briefly walk through each of these aspects. Along the way, I will point to parts of an example model. The model describes how a car transports a passenger.

We begin with the structure of a system. In SBF, the structure of a system is denoted by components and the attributes of those components. A component denotes a physical part of the system. For example, Car and Wheels are components of this system. An attribute is either Quantitative, meaning its values are numbers, or Descriptive, meaning its values are words. For example, Velocity is a Quantitative attribute of Car, and Engaged is a Descriptive attribute of Brakes.

Do you have any questions about modeling structure?

[Pause to answer questions about the structure model]

Next, the functions of a system. A function is an objective or goal of the system or some part of it. A function may either come from the system's designer or, especially when talking about biological systems, it may come from our interpretation. For example, the function of a car may be that a Car Transports a Passenger. Each function points to a behavior that achieves it and a set of results– values of a component attributes– that must be true at the completion of the function. For example, one result for Car Transports a Passenger is that the Passenger's Distance Travelled is 10.0.

A function may have subfunctions that help to achieve it. For example, Engine Turns Wheels and Brakes Stop Car. In the experiment, I will give you a page that shows all the expected function-subfunction relationships for the system you will be modeling.

Do you have any questions about modeling functions?

[Pause to answer questions about functions]

Finally, the behaviors of a system. You will be modeling behaviors in this experiment, so this portion is especially important. A behavior describes a process that the system goes through to achieve one of the its functions.

A behavior consists of a linear sequence of states. Each state is a snapshot of a moment in time, which is expressed as a set of attribute values. A behavior must have at least two states, a Start state representing the beginning of the process and a Stop state representing the end of it; but it can have more if needed. For example, the Start State of Car Transports a Passenger Behavior describes how the distance travelled of the passenger is 0.0, the car's velocity is 0.0, and so on. The state after that describes that the Engine's State is now On.

Transitions exist between states and describe the process moving from one state to the next. A transition should be annotated with one or more explanations that say why that move happens. Each explanation describes why and/or how attribute values change. There are four types of explanations: principle, stimulus, function, and equation.

[pause for questions]

A **principle explanation** says that the transition happens because of some physical principle. For example, in Transition-2 of Engine Turns Wheels Behavior, we say that Traction_of_the_tires_against_the_street is why the car's velocity increases. When the wheels turn, the velocity increases because the wheels grip the road and propel the car forward.

A **stimulus explanation** says that the transition happens because some entity not part of the system changed something in the system. For example, in Transition-1 of Car Transports a Passenger Behavior, we say that the Engine's state becomes On because the driver– which we have chosen to be external to the system– turns it on.

A **function explanation** says that the transition happens because of some subfunction. For example, in Transition-2 of Car Transports Passenger Behavior, we say that this transition happens because of the subfunction Engine Turns Wheels.

Finally, an **equation explanation** says what an attribute's value will be in the next state based on some formula, which may contain references to other attributes. For example, in Transition-3 of Car Transports Passenger Behavior, the equation relates to Passenger's new Distance Travelled value to be her old Distance Travelled plus the velocity of the Car.

Formulas present in principle, stimulus, and equation explanations have a particular form or syntax, and I will give you more details about that in a moment.

Do you have any questions about modeling behaviors?

[Pause to answer questions about behaviors]

Since you will be creating behaviors as part of this experiment, I would like to teach you how to use the modeling interface. Let us try to build part of a behavior together. First, we will erase most of the Car Transports a Passenger Behavior.

[Show how to delete conditions, explanations, and states]

[Show how to add states, add conditions, and add explanations. Have the participant rebuild part of the behavior you just deleted. Use the last page of this document to help.]

Do you have any questions about using the modeling interface?

[Pause to answer questions about using the modeling interface]

We have now talked about all three parts of an SBF model: the structure, the functions, and the behaviors. Please remember that you will be asked to model behaviors. Do you have any additional questions?

[Pause to answer questions]

[IF PARTICIPANT IS NOT IN EXPERIMENTAL CONDITION, SKIP TO "END SKIP HERE"]

While building a model in this experiment, you will have the ability to use automated verification technologies to check your model through simulation and model comparison. These technologies will appear in the right-side of the modeling interface if they are available to you. Simply press the Verify My Model button to ask the computer for its feedback. You may have to wait a moment to receive it. You may press this button as many times as you would like during the experiment.

[Show user what pressing the button does.]

When using the Model Comparison tab, you will see a link to view the model from the computer's knowledgebase. This opens an image in a new tab. The model in the computer's knowledgebase will always have the same structure and functions as your model, but its behaviors may be different. I will now show you now to read a behavior in this image.

[Show user how to read a behavior.]

Finally, I will give you some paper while you model. Please use this paper to write down when you choose to disregard something that the automated verification technologies tell you if you ever choose to do that. Please also write down a sentence or two about why you chose to do so. You are absolutely allowed to disregard things if you think that is appropriate. This paper will help me know that you saw what the automated verification technologies said and intentionally chose to do something else or not take action and why you chose to do this.

Do you have any questions about the automated verification technologies?

[Pause to answer questions about the verification technologies]

[END TO SKIP HERE]

[Hand out the equation explanation sheet.]

I have just given you a short guide on how to write formulas for equation, principle, and stimulus transition explanations. Formulas have a precise form or syntax because we want them to be readable by a computer. Please read through this handout and let me know when you are done. Note that you may keep this handout for the remainder of the experiment to refer back to it.

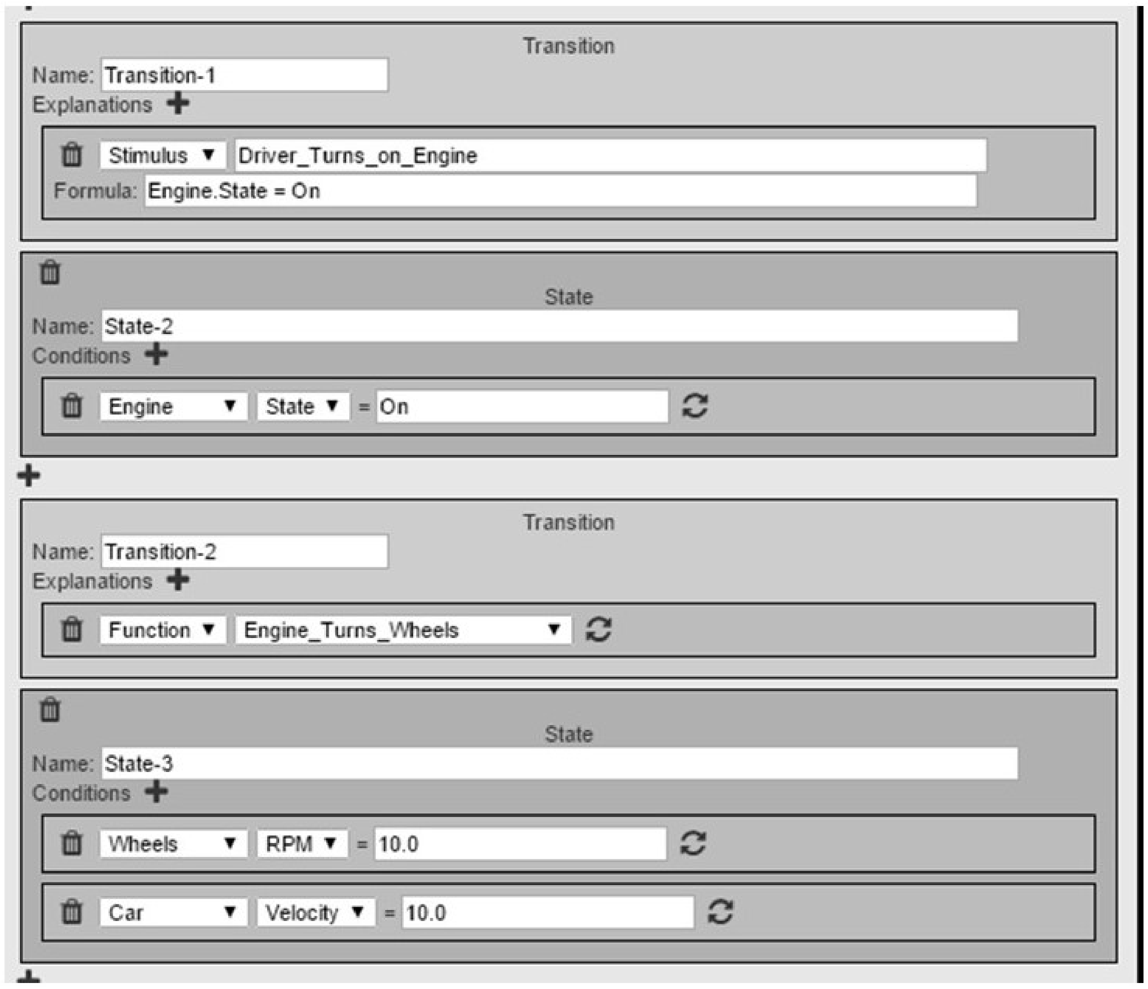
[Wait for them.]

Do you have any questions about what you just read?

[Pause to answer questions]

Participant has 45 minutes to make model.

Part of the Model for Reconstruction


Transition
Name: Transition-1
Explanations
Stimulus ▼ Driver_Turns_on_Engine
Formula: Engine.State = On

State
Name: State-2
Conditions
Engine ▼ State ▼ = On

Transition
Name: Transition-2
Explanations
Function ▼ Engine_Turns_Wheels ▼

State
Name: State-3
Conditions
Wheels ▼ RPM ▼ = 10.0
Car ▼ Velocity ▼ = 10.0

Formula Writing Guide

Below is the guide given to the participants about how to write formulas in the model interface during this experiment. All participants, regardless of condition, received this guide and received the same version of it.

Guide

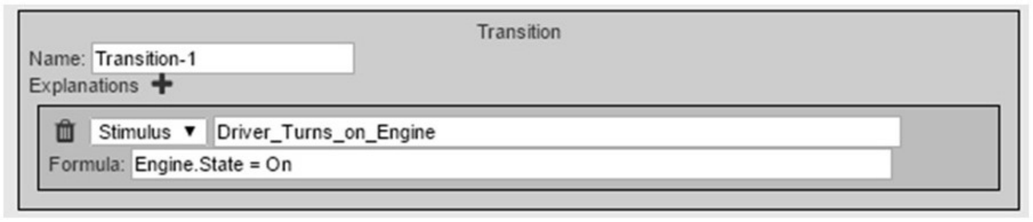
How to write formulas for this experiment

In transitions, the principle, stimulus, and equation explanation types can accept a formula to describe the change in an attribute's value because of that explanation. Formulas in this experiment have a precise form, or syntax, because we want them to be readable by a computer. This document will explain that syntax.

First, please note that although these formulas look very mathematical and may use numbers, the numbers do not necessarily need to capture precise, technical values. Instead, the numbers may refer to abstract amounts where we only really care about how they change over time (if they increase, decrease, or stay the same) and relate to each other.

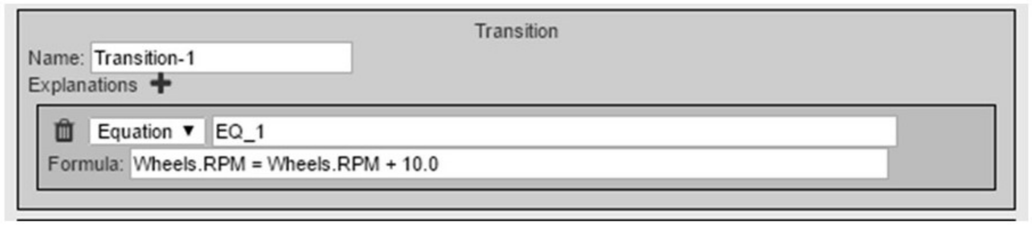
Below are two examples formulas from the Car Transports a passenger model.

(a)



Transition
Name: Transition-1
Explanations
Stimulus ▼ Driver_Turns_on_Engine
Formula: Engine.State = On

(b)



Transition
Name: Transition-1
Explanations
Equation ▼ EQ_1
Formula: Wheels.RPM = Wheels.RPM + 10.0

A formula assigns a value to the attribute written on the left-hand side of the equals sign to the result of the expression on the right-hand side of the equals sign. Attributes are written as Component.Attribute, meaning the Attribute of the Component such as the Engine's State in Engine.State and the Wheels' RPM in Wheels.RPM in our examples. The attribute will take on its assigned value in the state after the transition containing the formula. For example, in the next state, Engine.State will equal On and Wheels.RPM will equal its value in the previous state plus 10.0.

As in these examples, you write a formula in the Formula: field associated with an Equation, Stimulus, or (not pictured) Principle explanation. There must always be a formula for an Equation explanation, but it is optional for a Stimulus or Principle explanation. That said, a formula always needs to be provided if you want the explanation to express a change in an attribute value. Otherwise, the computer will not know how to interpret your explanation. This is true even if the explanation is a repeat of one elsewhere in the model.

Syntax of equation descriptions

Equation descriptions have the following syntax.

`<assigned-to-attribute> = <expression>`

`<assigned-to-attribute>` is the attribute that gets assigned the result of `<expression>`. It should be in the form Component.Attribute. For example, Engine.State in example (a) or Wheels.RPM in example (b).

When your `<assigned-to-attribute>` refers to a Descriptive attribute, such as in Engine.State in example (a), the computer will assign everything in `<expression>` to be the value of `<assigned-to-attribute>` in the next state. In (a), this means that the word On will be the value of the State of Engine (Engine.State) in the next state. Note that the computer will not do any interpretation of `<expression>` when assigning values to Descriptive attributes, so you cannot refer to other attributes here as in the instructions below.

When your `<assigned-to-attribute>` refers to a Quantitative attribute, such as in Wheels.RPM in example (b), `<expression>` is a mathematical expression. For operators in this expression, you may use addition (+), subtraction (-), multiplication (*), and division (/). You may also use parentheses to define order of operations. For values in this expression, you may use numbers and may reference other attributes (e.g., Car.Velocity). Only Quantitative attributes should be referenced. To reference an attribute, use the same Component.Attribute construction as in `<assigned-to-attribute>`. Attributes referenced in `<expression>` look to the state before the transition containing this formula for their values. For example, in (b) the Wheels.RPM appearing in the `<expression>` refers to the value of Wheels.RPM in the state before Transition-1.

Multiple Formulas in a Transition

Although it is not pictured here, multiple formulas may appear in a transition because a transition may have multiple explanations and you can have one formula per Equation, Principle, or Stimulus explanation. This is absolutely allowed. However, please note that formulas within a single transition cannot influence each other. Any attributes referenced in the <`expression`> part of the formulas will refer to attribute values in the state before their transition (i.e., the previous state), and the formulas will thus be unaware of any attribute values that will exist in the next state.

Therefore, if you wish to model the change in one attribute causing a change in another attribute, we recommend that you do this across two or more transitions.

Pre- and Post-Modeling Surveys

Below are the Modeling Surveys used in the experiment. There is one for the Biological condition and one for the Technological condition. Both the Pre-Survey (given to a participant before getting the Source Materials, getting the Expected Function-Subfunction Relationships, and constructing a model) and the Post-Survey (given to a participant after they finished model-building) were identical within the same condition, so only one version of each survey is given.

Biological Condition

1. How well do you understand the system described in the Modeling Prompt: friction drag (a.k.a., viscous drag) generated by a flat plate moving through a fluid? Please select the number that best reflects your answer.

I do not understand it at all						I understand it completely
1	2	3	4	5	6	7

2. As best as you can, please describe the system in the Modeling Prompt: friction drag (a.k.a., viscous drag) generated by a flat plate moving through a fluid. If you chose 1

(I do not understand it at all) for the previous question, you may skip this question. Please ask for more paper if you need more room to write.

Technological Condition

1. How well do you understand the system described in the Modeling Prompt: an electric toothbrush that cleans teeth by rotating its brush head clockwise and counterclockwise? Please select the number that best reflects your answer.

I do not understand it at all						I understand it completely
1	2	3	4	5	6	7

2. As best as you can, please describe the system in the Modeling Prompt: an electric toothbrush that cleans teeth by rotating its brush head clockwise and counterclockwise. If you chose 1 (I do not understand it at all) for the previous question, you may skip this question. Please ask for more paper if you need more room to write.

Modeling Prompts

Below are the two modeling prompts used in the experiment. One was given to participants in the Biological condition, and the other was given to participants in the Technological condition.

Biological Condition Prompt

Biologically inspired designers look to nature for inspiration to address design challenges. Shark skin is one such source of inspiration. It reduces the amount of drag that the shark generates while swimming underwater.

Rather than modeling how the shark skin achieves this, we want you to model a more fundamental phenomenon. We would you like to model how friction drag (a.k.a., viscous drag) gets generated as a flat plate moves through a fluid. Note that friction or viscous drag is not the only kind of drag that gets generated in this situation, but we want you to focus solely on this kind of drag.

If you have access to the model verification technology, please note that the model in the computer's knowledgebase that it will use to compare against your model is at a conceptual level and at a very high level of abstraction. In it, lots of details are left out, and the numbers in the model are used solely to show how values increase, decrease, and relate to each other. The numbers do not reflect precise quantities. You may also use numbers in this way in your model, but that is not required.

Technological Condition Prompt

An electric toothbrush is a device that people use to clean their teeth. In this experiment, we would like you to model how an electric toothbrush cleans teeth by turning its brush head clockwise and counterclockwise. As part of this, please include how the electric toothbrush moves its head clockwise and counterclockwise.

If you have access to the model verification technology, please note that the model in the computer's knowledgebase that it will use to compare against your model uses numbers solely to show how values increase, decrease, and relate to each other. The numbers do not reflect precise quantities. You may also use numbers in this way in your model, but that is not required.

Expected Function-Subfunction Relationships

Below are the Expected Function-Subfunction Relationships handouts given to participants at the beginning of their model-building time. One version was given to Biological condition participants, and another version was given to Technological condition participants.

Biological Condition

Function	**Subfunction**
Flat_Plate_in_Fluid_Generates_Drag	Momentum_Transfer_Generates_Drag
Momentum_Transfer_Generates_Drag	Momentum_Is_Transferred_Due _to_Bursting_Vortices
Momentum_Is_Transferred_Due _to_Bursting_Vortices	*none*

Technological Condition

Function	**Subfunction(s) (in alphabetical order)**
Clean_Teeth	Convert_Continuous_Mechanical_Rotational_Energy _to_Intermittent_Rotational_Movement
	Convert_Electrical_Energy_to_Continuous _Mechanical_Rotational_Energy
	Move_Head_Clockwise
	Move_Head_Counterclockwise
	Turn_Off_Power
	Turn_On_Power

Note: **Clean_Teeth is the only function with expected subfunctions for this system.**

Source Materials

Participants were given Source Materials to use in constructing a model of their condition's topic. One version of the Source Materials was given to Biological condition participants, and another version was given to Technological condition participants.

For copyright reasons, the Source Materials are not recreated here. Instead, a brief description is provided along with a reference to where the reader can find the materials on their own.

Biological Condition

The Biological condition Source Material was part of a scientific paper [38] on shark skin. In particular, the material given to participants was pages 4776-4779 with parts at the beginning and ending hidden to simplify the document. These pages mostly described how fluid drag (including friction drag) works. The document provided both textual descriptions and images to help the participant reason about the topic.

Technological Condition

The Technological condition Source Material was part of a printed out[2] website [173] that described how electric toothbrushes worked. The website provided both textual descriptions and images that participants could use to understand the topic. Participants were shown content up until (but not including) the "What's the difference between a sonic toothbrush and an ordinary electric one?" part of the website that existed at the time of the printout. This worked out to about four and a half pages of content.

Exit Survey

Below is the Exit Survey given to participants. One version of the Exit Survey was given to Control condition participants, and another version was given to Experimental condition participants. Note that the first question in both surveys was intended to be identical but there is an accidental one word difference between them (technology versus interface).

Control Condition

1. Please describe at least 3 ways that you would improve the model building technology that you used.

[2]Remote participants received a file containing the printed out version, so even though they didn't have a physical copy like local participants, the content and presentation between the two situations were identical.

Experimental Condition

1. Please describe at least 3 ways that you would improve the model building interface that you used.

2. To what extent did the automated verification technology impact your model building? If you built more than one model, please consider all models that you built. Please select the number that best reflects your answer.

Very Negatively Impacted			No Impact			Very Positively Impacted
-3	-2	-1	0	1	2	3

3. Unless you answered 0 (No Impact) for question 2, please describe at least 3 ways that the automated verification technology impacted your model building. You may describe both positive and negative impacts regardless of your answer to question 2.

4. To what extent did the automated verification technology impact your understanding of the topic that you modeled? Please select the number that best reflects your answer.

Very Negatively Impacted			No Impact			Very Positively Impacted
-3	-2	-1	0	1	2	3

5. Please describe at least 3 ways that you would improve the automated verification technology.

APPENDIX D

SBF* MODELS IN DISSERTATION

This appendix presents the Structure-Behavior-Function Star (SBF*) models developed for and used in this dissertation along with brief descriptions of their topics and diagrams of their functional decompositions to aid in understanding. These models are all in the Original model configuration as per the Computational Experimentation (see Chapter 5), meaning that they are in a form intended to accurately represent their modeling topic, do not have any intentional ablations or modifications, and all will pass the Simulation technique of the Design Evaluation through Simulation and Comparison (DESC) AI agent without issues. For models that also appeared in the Usefulness Study (Chapter 6), these models should be identical to those that were in the computer's knowledge base[1] except that their names (e.g., `Original_ExpertTechnologicalModel` for Electric Toothbrush's model) may have been changed.

Each model is presented in the textual version of the SBF* language described in Chapter 3 and whose language definition is in Appendix A. Comments have been removed and the contents have been formatted but not meaningfully changed. The models are presented in alphabetical order by their code name. The code names used here for the models (e.g., Electric Toothbrush) are also used in the rest of this dissertation to aid cross-referencing between the body of the document and this appendix.

[1]This is believed to be accurate, but there is a possibility that they differ due to organizing issues.

D.1 Electric Toothbrush

D.1.1 Description and Functional Decomposition

This model was derived from a website article by Woodford [173]. Note that the currently available version of these source materials appears to be a newer version than what was used to develop this model and what was used by the Usefulness Study participants.

This model describes how the head of an electric toothbrush rotates clockwise and counterclockwise to reduce the amount of bacteria on teeth. To do so, once the toothbrush is turned on, the rechargeable battery supplies electrical energy that gets converted into first continuous rotational energy and then intermittent rotational movement. This results in the brush head rotating one way and then back to its original orientation before the toothbrush gets turned off. As the brush head rotates, the amount of bacteria on teeth is reduced. Note that the quantities present in this model are abstract amounts and not intended to reflect precise real-world values.

Due to the complexity of this model and space limitations, the functional decomposition is broken into two figures. Figure D.1 diagrammatically presents the set of functions (white boxes) and the names of the behaviors that achieve them (black rounded boxes). Within each function box is its name and set of provides conditions. Figure D.2 presents the decomposition of these functions, with arrows going from the superfunction to the subfunction. In this case, `Clean_Teeth` is the sole superfunction and all others are subfunctions of it. The height of the subfunctions is not meaningful, but they are ordered from left to right in the order in which they appear in `Clean_Teeth`'s associated behavior.

D.1.2 The Model

```
Original_ExpertTechnologicalModel is an SBF model
 The model has a structure
  Teeth is a component
   It has a connecting point named C1
```

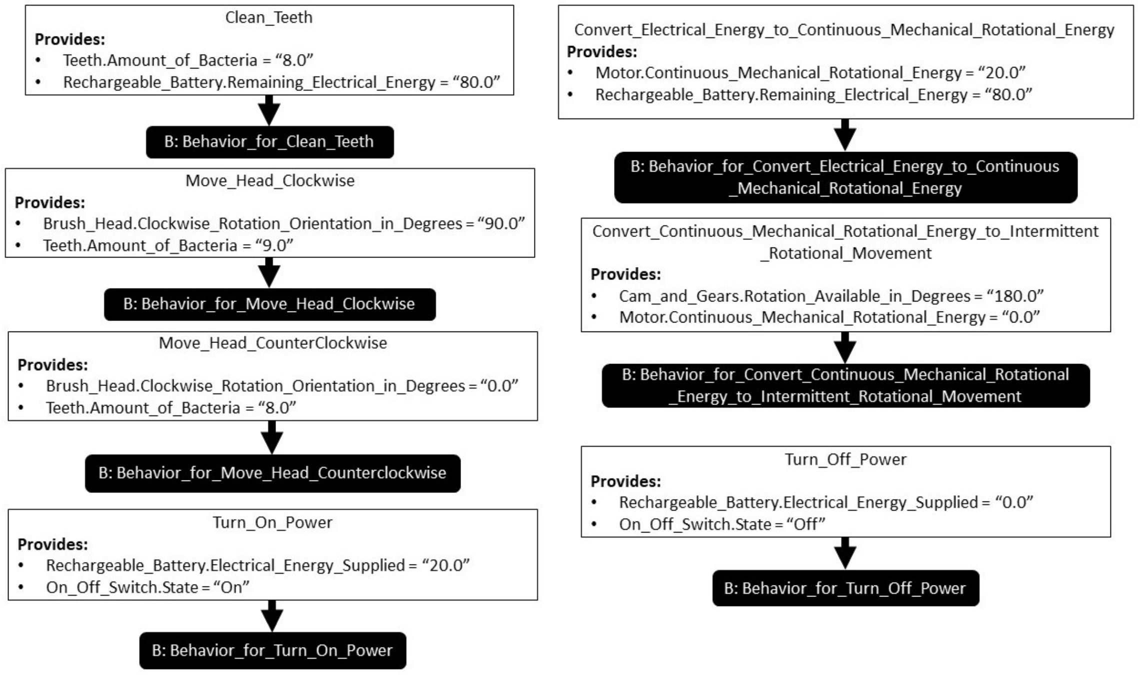

Clean_Teeth
Provides:
• Teeth.Amount_of_Bacteria = "8.0"
• Rechargeable_Battery.Remaining_Electrical_Energy = "80.0"
B: Behavior_for_Clean_Teeth
Move_Head_Clockwise
Provides:
• Brush_Head.Clockwise_Rotation_Orientation_in_Degrees = "90.0"
• Teeth.Amount_of_Bacteria = "9.0"
B: Behavior_for_Move_Head_Clockwise
Move_Head_CounterClockwise
Provides:
• Brush_Head.Clockwise_Rotation_Orientation_in_Degrees = "0.0"
• Teeth.Amount_of_Bacteria = "8.0"
B: Behavior_for_Move_Head_Counterclockwise
Turn_On_Power
Provides:
• Rechargeable_Battery.Electrical_Energy_Supplied = "20.0"
• On_Off_Switch.State = "On"
B: Behavior_for_Turn_On_Power
Convert_Electrical_Energy_to_Continuous_Mechanical_Rotational_Energy
Provides:
• Motor.Continuous_Mechanical_Rotational_Energy = "20.0"
• Rechargeable_Battery.Remaining_Electrical_Energy = "80.0"
B: Behavior_for_Convert_Electrical_Energy_to_Continuous _Mechanical_Rotational_Energy
Convert_Continuous_Mechanical_Rotational_Energy_to_Intermittent _Rotational_Movement
Provides:
• Cam_and_Gears.Rotation_Available_in_Degrees = "180.0"
• Motor.Continuous_Mechanical_Rotational_Energy = "0.0"
B: Behavior_for_Convert_Continuous_Mechanical_Rotational _Energy_to_Intermittent_Rotational_Movement
Turn_Off_Power
Provides:
• Rechargeable_Battery.Electrical_Energy_Supplied = "0.0"
• On_Off_Switch.State = "Off"
B: Behavior_for_Turn_Off_Power

Figure D.1: Functions in the Electric Toothbrush model and their associated behaviors

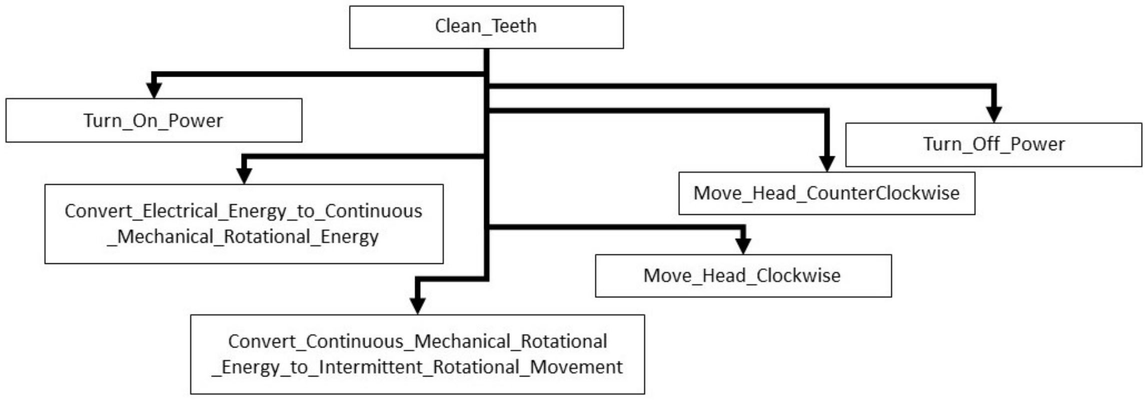

Clean_Teeth
Turn_On_Power
Turn_Off_Power
Move_Head_CounterClockwise
Convert_Electrical_Energy_to_Continuous _Mechanical_Rotational_Energy
Move_Head_Clockwise
Convert_Continuous_Mechanical_Rotational _Energy_to_Intermittent_Rotational_Movement

Figure D.2: Functional decomposition of the Electric Toothbrush model

```
  It has an attribute Amount_of_Bacteria of type Quantitative

On_Off_Switch is a component
  It has a connecting point named C1
  It has an attribute State of type Descriptive

Motor is a component
  It has a connecting point named C1
  It has an attribute Continuous_Mechanical_Rotational_Energy
   of type Quantitative
```

```
Rechargeable_Battery is a component
 It has a connecting point named C1
 It has an attribute Electrical_Energy_Supplied of type
  Quantitative
 It has an attribute Remaining_Electrical_Energy of type
  Quantitative

Cam_and_Gears is a component
 It has a connecting point named C1
 It has an attribute Rotation_Available_in_Degrees of type
  Quantitative

Brush_Head is a component
 It has a connecting point named C1
 It has an attribute Clockwise_Rotation_Orientation_in_Degrees
  of type Quantitative

The model has behaviors

 Behavior_for_Clean_Teeth is a behavior

  Start_State is the start state
   It has the condition: (
    Teeth.Amount_of_Bacteria = "10.0"
    and On_Off_Switch.State = "Off"
    and Rechargeable_Battery.Electrical_Energy_Supplied = "0.0"
    and Rechargeable_Battery.Remaining_Electrical_Energy =
     "100.0"
    and Cam_and_Gears.Rotation_Available_in_Degrees = "0.0"
    and Brush_Head.Clockwise_Rotation_Orientation_in_Degrees =
     "0.0"
  )

  State_2 is a state
   It has the condition: (
    On_Off_Switch.State = "On"
    and Rechargeable_Battery.Electrical_Energy_Supplied = "20.0"
   )

  State_3 is a state
   It has the condition: (
    Rechargeable_Battery.Remaining_Electrical_Energy = "80.0"
    and Motor.Continuous_Mechanical_Rotational_Energy = "20.0"
   )

  State_4 is a state
   It has the condition: (
    Motor.Continuous_Mechanical_Rotational_Energy = "0.0"
    and Cam_and_Gears.Rotation_Available_in_Degrees = "180.0"
   )

  State_5 is a state
   It has the condition: (
    Teeth.Amount_of_Bacteria = "9.0"
    and Cam_and_Gears.Rotation_Available_in_Degrees = "90.0"
    and Brush_Head.Clockwise_Rotation_Orientation_in_Degrees =
     "90.0"
   )

  State_6 is a state
   It has the condition: (
    Teeth.Amount_of_Bacteria = "8.0"
    and Cam_and_Gears.Rotation_Available_in_Degrees = "0.0"
    and Brush_Head.Clockwise_Rotation_Orientation_in_Degrees =
     "0.0"
   )
```

```
    Stop_State is the stop state
     It has the condition: (
      On_Off_Switch.State = "Off"
      and Rechargeable_Battery.Electrical_Energy_Supplied = "0.0"
     )

    Transition_1 is a transition from Start_State to State_2
     It has the explanation (
      the function Turn_On_Power
     )

    Transition_2 is a transition from State_2 to State_3
     It has the explanation (
      the function Convert_Electrical_Energy_to_Continuous
       _Mechanical_Rotational_Energy
     )

    Transition_3 is a transition from State_3 to State_4
      It has the explanation (
       the function Convert_Continuous_Mechanical_Rotational
        _Energy_to_Intermittent_Rotational_Movement
      )

    Transition_4 is a transition from State_4 to State_5
     It has the explanation (
      the function Move_Head_Clockwise
     )

    Transition_5 is a transition from State_5 to State_6
     It has the explanation (
      the function Move_Head_Counterclockwise
     )

    Transition_6 is a transition from State_6 to Stop_State
     It has the explanation (
      the function Turn_Off_Power
     )

   Behavior_for_Move_Head_Clockwise is a behavior

    Start_State is the start state
     It has the condition: (
      Teeth.Amount_of_Bacteria = "10.0"
      and Brush_Head.Clockwise_Rotation_Orientation_in_Degrees =
       "0.0"
      and Cam_and_Gears.Rotation_Available_in_Degrees = "180.0"
     )

    State_2 is a state
     It has the condition: (
      Brush_Head.Clockwise_Rotation_Orientation_in_Degrees =
       "90.0"
      and Cam_and_Gears.Rotation_Available_in_Degrees = "90.0"
     )

    Stop_State is the stop state
     It has the condition: (
      Teeth.Amount_of_Bacteria = "9.0"
     )

    Transition_1 is a transition from Start_State to State_2
     It has the explanation (
      the equation Brush_Head_Rotates_to_New_Orientation,
       described as "eq:
       Brush_Head.Clockwise_Rotation_Orientation_in_Degrees =
      Brush_Head.Clockwise_Rotation_Orientation_in_Degrees:Before
      + 90.0"
     and the equation Available_Intermittent_Rotational_Movement
      _is_Used_to_Rotate_Brush_Head, described as "eq:
      Cam_and_Gears.Rotation_Available_in_Degrees
      = Cam_and_Gears.Rotation_Available_in_Degrees:Before -
      90.0"
    )

  Transition_2 is a transition from State_2 to Stop_State
    It has the explanation (
     the principle Rotating_Brush_Head_Removes_Bacteria,
      described as "eq: Teeth.Amount_of_Bacteria =
      Teeth.Amount_of_Bacteria:Before - 1.0"
    )

Behavior_for_Move_Head_Counterclockwise is a behavior

  Start_State is the start state
    It has the condition: (
     Teeth.Amount_of_Bacteria = "9.0"
     and Brush_Head.Clockwise_Rotation_Orientation_in_Degrees =
      "90.0"
     and Cam_and_Gears.Rotation_Available_in_Degrees = "90.0"
    )

  State_2 is a state
    It has the condition: (
     Brush_Head.Clockwise_Rotation_Orientation_in_Degrees = "0.0"
     and Cam_and_Gears.Rotation_Available_in_Degrees = "0.0"
    )

  Stop_State is the stop state
    It has the condition: (
     Teeth.Amount_of_Bacteria = "8.0"
    )

  Transition_1 is a transition from Start_State to State_2
    It has the explanation (
     the equation Brush_Head_Rotates_Back_to_Original
      _Orientation, described as "eq:
      Brush_Head.Clockwise_Rotation_Orientation_in_Degrees =
      Brush_Head.Clockwise_Rotation_Orientation_in_Degrees:Before
      - 90.0"
     and the equation Available_Intermittent_Rotational_Movement
      _is_Used_to_Rotate_Brush_Head, described as "eq:
      Cam_and_Gears.Rotation_Available_in_Degrees
      = Cam_and_Gears.Rotation_Available_in_Degrees:Before -
      90.0"
    )

  Transition_2 is a transition from State_2 to Stop_State
    It has the explanation (
     the principle Rotating_Brush_Head_Removes_Bacteria,
      described as "eq: Teeth.Amount_of_Bacteria =
      Teeth.Amount_of_Bacteria:Before - 1.0"
    )

Behavior_for_Convert_Electrical_Energy_to_Continuous
 _Mechanical_Rotational_Energy is a behavior

  Start_State is the start state
    It has the condition: (
     Rechargeable_Battery.Electrical_Energy_Supplied = "20.0"
     and Rechargeable_Battery.Remaining_Electrical_Energy =
        "100.0"
      and Motor.Continuous_Mechanical_Rotational_Energy = "0.0"
     )

    Stop_State is the stop state
     It has the condition: (
      Rechargeable_Battery.Remaining_Electrical_Energy = "80.0"
      and Motor.Continuous_Mechanical_Rotational_Energy = "20.0"
     )

    Transition_1 is a transition from Start_State to Stop_State
     It has the explanation (
      the equation Electrical_Energy_is_Converted_into_Continuous
       _Mechanical_Rotational_Energy_Part_1, described as "eq:
       Motor.Continuous_Mechanical_Rotational_Energy =
       Motor.Continuous_Mechanical_Rotational_Energy:Before +
       Rechargeable_Battery.Electrical_Energy_Supplied:Before"
      and the equation Electrical_Energy_is_Converted_into
       _Continuous_Mechanical_Rotational_Energy_Part_2,
       described as "eq:
       Rechargeable_Battery.Remaining_Electrical_Energy =
       Rechargeable_Battery.Remaining_Electrical_Energy:Before -
       Rechargeable_Battery.Electrical_Energy_Supplied:Before"
     )

   Behavior_for_Convert_Continuous_Mechanical_Rotational_Energy
    _to_Intermittent_Rotational_Movement is a behavior

    Start_State is the start state
     It has the condition: (
      Cam_and_Gears.Rotation_Available_in_Degrees = "0.0"
      and Motor.Continuous_Mechanical_Rotational_Energy = "20.0"
     )

    Stop_State is the stop state
     It has the condition: (
      Cam_and_Gears.Rotation_Available_in_Degrees = "180.0"
      and Motor.Continuous_Mechanical_Rotational_Energy = "0.0"
     )

    Transition_1 is a transition from Start_State to Stop_State
     It has the explanation (
      the equation Continuous_Mechanical_Rotational_Energy_is
       _Converted_into_Available_Intermittent_Rotational
       _Movement_Part_1, described as "eq:
       Cam_and_Gears.Rotation_Available_in_Degrees =
       Cam_and_Gears.Rotation_Available_in_Degrees:Before
       + 180.0"
      and the equation Continuous_Mechanical_Rotational_Energy_is
       _Converted_into_Available_Intermittent_Rotational
       _Movement_Part_2, described as "eq:
       Motor.Continuous_Mechanical_Rotational_Energy =
       Motor.Continuous_Mechanical_Rotational_Energy:Before -
       20.0"
     )

   Behavior_for_Turn_On_Power is a behavior

    Start_State is the start state
     It has the condition: (
      On_Off_Switch.State = "Off"
      and Rechargeable_Battery.Electrical_Energy_Supplied = "0.0"
     )

    State_2 is a state
```

```
     It has the condition: (
      On_Off_Switch.State = "On"
     )

    Stop_State is the stop state
     It has the condition: (
      Rechargeable_Battery.Electrical_Energy_Supplied = "20.0"
     )

    Transition_1 is a transition from Start_State to State_2
     It has the explanation (
      the stimulus User_Presses_Switch, described as "eq:
       On_Off_Switch.State = On"
     )

    Transition_2 is a transition from State_2 to Stop_State
     It has the explanation (
      the principle On_Switch_Allows_Electrical_Energy_To_Flow,
       described as "eq:
       Rechargeable_Battery.Electrical_Energy_Supplied = 20.0"
     )

   Behavior_for_Turn_Off_Power is a behavior

    Start_State is the start state
     It has the condition: (
      On_Off_Switch.State = "On"
      and Rechargeable_Battery.Electrical_Energy_Supplied = "20.0"
     )

    State_2 is a state
     It has the condition: (
      On_Off_Switch.State = "Off"
     )

    Stop_State is the stop state
     It has the condition: (
      Rechargeable_Battery.Electrical_Energy_Supplied = "0.0"
     )

    Transition_1 is a transition from Start_State to State_2
     It has the explanation (
      the stimulus User_Presses_Switch, described as "eq:
       On_Off_Switch.State = Off"
     )

    Transition_2 is a transition from State_2 to Stop_State
     It has the explanation (
      the principle Off_Switch_Prevents_Electrical_Energy
       _From_Flowing, described as "eq:
       Rechargeable_Battery.Electrical_Energy_Supplied = 0.0"
     )

  The model has functions

   Clean_Teeth is a function
    It requires nothing
    It is achieved by the behavior Behavior_for_Clean_Teeth
    It provides: (
     Teeth.Amount_of_Bacteria = "8.0"
     and Rechargeable_Battery.Remaining_Electrical_Energy = "80.0"
    )

   Move_Head_Clockwise is a function
    It requires nothing
    It is achieved by the behavior
     Behavior_for_Move_Head_Clockwise
    It provides: (
     Brush_Head.Clockwise_Rotation_Orientation_in_Degrees = "90.0"
     and Teeth.Amount_of_Bacteria = "9.0"
    )

   Move_Head_Counterclockwise is a function
    It requires nothing
    It is achieved by the behavior
     Behavior_for_Move_Head_Counterclockwise
    It provides: (
     Brush_Head.Clockwise_Rotation_Orientation_in_Degrees = "0.0"
     and Teeth.Amount_of_Bacteria = "8.0"
    )

   Convert_Electrical_Energy_to_Continuous_Mechanical_Rotational
    _Energy is a function
    It requires nothing
    It is achieved by the behavior
     Behavior_for_Convert_Electrical_Energy_to_Continuous
     _Mechanical_Rotational_Energy
    It provides: (
     Motor.Continuous_Mechanical_Rotational_Energy = "20.0"
     and Rechargeable_Battery.Remaining_Electrical_Energy = "80.0"
    )

   Convert_Continuous_Mechanical_Rotational_Energy_to_Intermittent
    _Rotational_Movement is a function
    It requires nothing
    It is achieved by the behavior
     Behavior_for_Convert_Continuous_Mechanical_Rotational
     _Energy_to_Intermittent_Rotational_Movement
    It provides: (
     Cam_and_Gears.Rotation_Available_in_Degrees = "180.0"
     and Motor.Continuous_Mechanical_Rotational_Energy = "0.0"
    )

   Turn_On_Power is a function
    It requires nothing
    It is achieved by the behavior Behavior_for_Turn_On_Power
    It provides: (
     Rechargeable_Battery.Electrical_Energy_Supplied = "20.0"
     and On_Off_Switch.State = "On"
    )

   Turn_Off_Power is a function
    It requires nothing
    It is achieved by the behavior Behavior_for_Turn_Off_Power
    It provides: (
     Rechargeable_Battery.Electrical_Energy_Supplied = "0.0"
     and On_Off_Switch.State = "Off"
    )
```

D.2 Friction Drag

D.2.1 Description and Functional Decomposition

This model was derived from [38]. This model describes the phenomenon of friction drag occurring with a flat plate in a fluid. Starting with a non-moving flat plate in a fluid, some external force causes the plate to start moving through the fluid, which gives it both relative velocity and momentum. The model then calculates the amount of momentum that will be transferred to the fluid. The amount of momentum transferred is related to bursting vortices and to a combination of the relative velocity of the flat plate, dynamic viscosity of the fluid, and a coefficient in the interface between the flat plate and the fluid. This momentum is lost from the flat plate and will go to the fluid. Drag is the amount of energy required to transfer this momentum, and this is calculated. Additionally, some of the momentum to be transferred is lost as it is converted into heat. After all of these calculations, the model describes the remaining total momentum being transferred to the fluid.

Figure D.3 diagrammatically presents the functional decomposition of the Friction Drag model. White boxes represent the functions, and behaviors that achieve them are black rounded boxes. Within each function box is its name and set of provides conditions. Arrows are drawn from functions to behaviors to illustrate that association. Arrows from behaviors to functions represent the function-subfunction relationship with the origin extending from the superbehavior (and thus superfunction) and head of the arrow pointing to the subfunction. In this case, `Flat_Plate_in_Fluid_Generates_Drag` is the topmost superfunction.

D.2.2 The Model

```
Original_ExpertBiologicalModel is an SBF model
 The model has a structure
  Flat_Plate is a component
   It has a connecting point named C1
   It has an attribute Drag of type Quantitative
   It has an attribute Momentum of type Quantitative
   It has an attribute Momentum_Transferred of type
    Quantitative
   It has an attribute Relative_Velocity of type
    Quantitative
   It has an attribute Mass of type Quantitative
  Flat_Plate_and_Fluid_Interface is a component
   It has a connecting point named C1
   It has an attribute Coefficient_for_Energy_Required_to
    _Transfer_Momentum of type Quantitative
   It has an attribute Coefficient_for_Heat_Generated_from
```

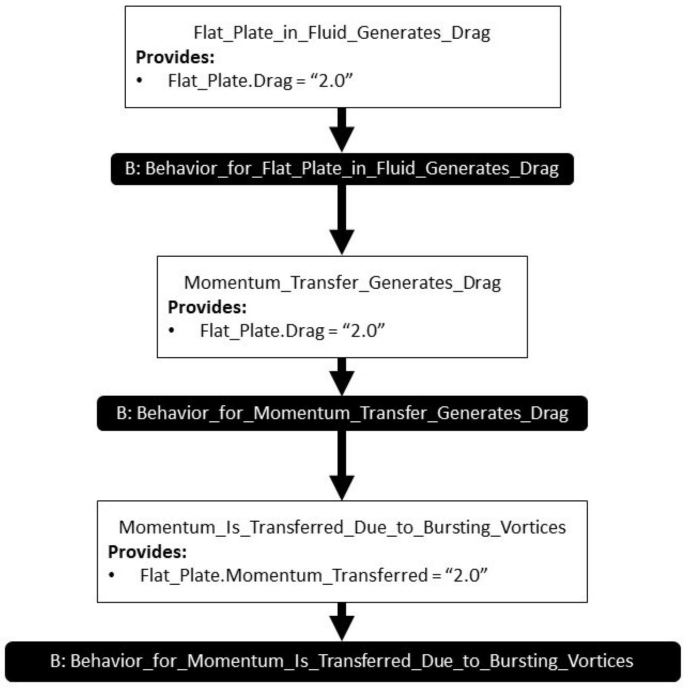

Flat_Plate_in_Fluid_Generates_Drag
Provides:
Flat_Plate.Drag = "2.0"
B: Behavior_for_Flat_Plate_in_Fluid_Generates_Drag
Momentum_Transfer_Generates_Drag
Provides:
Flat_Plate.Drag = "2.0"
B: Behavior_for_Momentum_Transfer_Generates_Drag
Momentum_Is_Transferred_Due_to_Bursting_Vortices
Provides:
Flat_Plate.Momentum_Transferred = "2.0"
B: Behavior_for_Momentum_Is_Transferred_Due_to_Bursting_Vortices

Figure D.3: Functional decomposition of the Friction Drag model with mediating behaviors

```
      _Interaction of type Quantitative
     It has an attribute Coefficient_for_Amount_of_Bursting
      _Vortices_Created of type Quantitative
     It has an attribute Coefficient_for_Amount_of_Momentum
      _Transferred of type Quantitative

    Fluid is a component
     It has a connecting point named C1
     It has an attribute Momentum of type Quantitative
     It has an attribute Dynamic_Viscosity of type Quantitative

    Bursting_Vortices is a component
     It has a connecting point named C1
     It has an attribute Amount of type Quantitative
     It has an attribute Momentum_Transferred_Per_Burst_Vortex
      of type Quantitative

    Generated_Heat is a component
     It has a connecting point named C1
     It has an attribute Energy_Amount of type Quantitative

   The model has behaviors

    Behavior_for_Flat_Plate_in_Fluid_Generates_Drag is a
     behavior

     Start_State is the start state
      It has the condition: (
       Flat_Plate.Drag = "0.0"
       and Flat_Plate.Momentum = "0.0"
       and Flat_Plate.Relative_Velocity = "0.0"
       and Fluid.Momentum = "0.0"
       and Flat_Plate.Mass = "10.0"
     )

     State_2 is a state
      It has the condition: (
       Flat_Plate.Relative_Velocity = "1.0"
      )

     State_3 is a state
      It has the condition: (
       Flat_Plate.Momentum = "10.0"
      )

     Stop_State is the stop state
      It has the condition: (
       Flat_Plate.Momentum = "8.0"
       and Fluid.Momentum = "1.5"
       and Flat_Plate.Drag = "2.0"
       and Generated_Heat.Energy_Amount = "0.5"
      )

     Transition_1 is a transition from Start_State to State_2
      It has the explanation (
       the stimulus External_force_moves_flat_plate, described
        as "eq: Flat_Plate.Relative_Velocity =
        Flat_Plate.Relative_Velocity:Before + 1.0"
      )

     Transition_2 is a transition from State_2 to State_3
      It has the explanation (
       the principle Momentum_is_proportional_to_mass_and
        _velocity, described as "eq: Flat_Plate.Momentum =
        Flat_Plate.Momentum:Before + Flat_Plate.Mass:Before *
        Flat_Plate.Relative_Velocity:Before"
      )

     Transition_3 is a transition from State_3 to Stop_State
      It has the explanation (
       the function Momentum_Transfer_Generates_Drag
      )

    Behavior_for_Momentum_Is_Transferred_Due_to_Bursting
     _Vortices is a behavior

     Start_State is the start state
      It has the condition: (
       Flat_Plate_and_Fluid_Interface.Coefficient_for_Amount
        _of_Bursting_Vortices_Created = "1.0"
       and Flat_Plate.Momentum_Transferred = "1.0"
       and Bursting_Vortices.Amount = "0.0"
       and Bursting_Vortices.Momentum_Transferred_Per_Burst
        _Vortex = "1.0"
      )

     State_2 is a state
      It has the condition: (
       Bursting_Vortices.Amount = "1.0"
      )

     Stop_State is the stop state
      It has the condition: (
       Flat_Plate.Momentum_Transferred = "2.0"
      )

     Transition_1 is a transition from Start_State to State_2
```

```
    It has the explanation (
     the equation Bursting_vortices_are_created, described as
      "eq: Bursting_Vortices.Amount =
      Bursting_Vortices.Amount:Before + Flat_Plate_and_Fluid
      _Interface.Coefficient_for_Amount_of_Bursting_Vortices
      _Created:Before"
    )

   Transition_2 is a transition from State_2 to Stop_State
    It has the explanation (
     the equation Momentum_is_transferred_due_to_burst
      _vortices, described as "eq:
      Flat_Plate.Momentum_Transferred =
      Flat_Plate.Momentum_Transferred:Before +
      Bursting_Vortices.Amount:Before *Bursting_Vortices.
      Momentum_Transferred_Per_Burst_Vortex:Before"
    )

  Behavior_for_Momentum_Transfer_Generates_Drag is a behavior

   Start_State is the start state
    It has the condition: (
     Flat_Plate.Drag = "0.0"
     and Flat_Plate.Momentum_Transferred = "0.0"
     and Flat_Plate.Momentum = "10.0"
     and Fluid.Momentum = "0.0"
     and Fluid.Dynamic_Viscosity = "1.0"
     and Flat_Plate_and_Fluid_Interface.Coefficient_for
      _Energy_Required_to_Transfer_Momentum = "1.0"
     and Generated_Heat.Energy_Amount = "0.0"
     and Flat_Plate_and_Fluid_Interface.Coefficient_for_Heat
      _Generated_from_Interaction = "0.5"
     and Flat_Plate.Relative_Velocity = "1.0"
     and Flat_Plate_and_Fluid_Interface.Coefficient_for
      _Amount_of_Momentum_Transferred = "1.0"
    )

   State_2 is a state
    It has the condition: (
     Flat_Plate.Momentum_Transferred = "1.0"
    )

   State_3 is a state
    It has the condition: (
     Flat_Plate.Momentum_Transferred = "2.0"
    )

   State_4 is a state
    It has the condition: (
     Flat_Plate.Momentum = "8.0"
    )

   State_5 is a state
    It has the condition: (
     Flat_Plate.Drag = "2.0"
    )

   State_6 is a state
    It has the condition: (
     Flat_Plate.Momentum_Transferred = "1.5"
     and Generated_Heat.Energy_Amount = "0.5"
    )

   State_7 is a state
    It has the condition: (
    Fluid.Momentum = "1.5"
   )

  Stop_State is the stop state
   It has the condition: (
    Flat_Plate.Momentum_Transferred = "0.0"
   )

  Transition_1 is a transition from Start_State to State_2
   It has the explanation (
    the equation Momentum_is_transferred, described as "eq:
      Flat_Plate.Momentum_Transferred = Flat_Plate.Momentum
      _Transferred:Before + Flat_Plate_and_Fluid_Interface.
      Coefficient_for_Amount_of_Momentum_Transferred:Before
      * Fluid.Dynamic_Viscosity:Before *
      Flat_Plate.Relative_Velocity:Before"
   )

  Transition_2 is a transition from State_2 to State_3
   It has the explanation (
    the function Momentum_Is_Transferred_Due_to_Bursting
      _Vortices
   )

  Transition_3 is a transition from State_3 to State_4
   It has the explanation (
    the equation Momentum_transferred_is_lost_from_the_Flat
      _Plate, described as "eq: Flat_Plate.Momentum =
      Flat_Plate.Momentum:Before -
      Flat_Plate.Momentum_Transferred:Before"
   )

  Transition_4 is a transition from State_4 to State_5
   It has the explanation (
    the principle Drag_is_energy_required_to_transfer
      _momentum, described as "eq: Flat_Plate.Drag =
      Flat_Plate.Drag:Before + Flat_Plate_and_Fluid
      _Interface.Coefficient_for_Energy_Required_to_Transfer
      _Momentum:Before * Flat_Plate.Momentum
      _Transferred:Before"
   )

  Transition_5 is a transition from State_5 to State_6
   It has the explanation (
    the principle Some_Transferred_Momentum_is_Not
      _Transferred_but_is_Actually_Generated_Heat, described
      as "eq: Flat_Plate.Momentum_Transferred = Flat_Plate.
      Momentum_Transferred:Before -  Flat_Plate_and_Fluid
      _Interface.Coefficient_for_Heat_Generated_from
      _Interaction:Before"
    and the equation Explanation-21, described as "eq:
      Generated_Heat.Energy_Amount = Generated_Heat.Energy
      _Amount:Before + Flat_Plate_and_Fluid_Interface.
      Coefficient_for_Heat_Generated_from_Interaction:Before"
   )

  Transition_6 is a transition from State_6 to State_7
   It has the explanation (
    the equation Momentum_transferred_to_the_fluid,
      described as "eq: Fluid.Momentum = Fluid.Momentum:
      Before + Flat_Plate.Momentum_Transferred:Before"
   )

  Transition_7 is a transition from State_7 to Stop_State
   It has the explanation (
    the equation Zero_out_momentum_transferred_to_show_it
      _has_now_all_been_transferred , described as "eq:
      Flat_Plate . Momentum_Transferred = 0.0"
   )

The model has functions

 Flat_Plate_in_Fluid_Generates_Drag is a function
  It requires nothing
  It is achieved by the behavior Behavior_for_Flat_Plate
   _in_Fluid_Generates_Drag
  It provides: (
   Flat_Plate . Drag = "2.0"
  )

 Momentum_Transfer_Generates_Drag is a function
  It requires nothing
  It is achieved by the behavior Behavior_for_Momentum
   _Transfer_Generates_Drag
  It provides: (
   Flat_Plate . Drag = "2.0"
  )

 Momentum_Is_Transferred_Due_to_Bursting_Vortices is a
  function
  It requires nothing
  It is achieved by the behavior Behavior_for_Momentum_Is
   _Transferred_Due_to_Bursting_Vortices
  It provides: (
   Flat_Plate . Momentum_Transferred = "2.0"
  )
```

D.3 Medical Patch

D.3.1 Description and Functional Decomposition

This model was derived from McKeag [101]. That article contains a brief section about a medical adhesive mechanism that was inspired by a mechanism of the spiny headed worm. The design works to secure a patch by inserting microneedles and, once inserted, expanding their tips to create fiction and hold the patch in place. The model thus describes a patch being inserted, the conical tips of the microneedle array (which is in the patch) expanding, and the patch resisting an applied pull force because of the friction created by the expanded tips.

Figure D.4 diagrammatically presents the functional decomposition of the Medical Patch model. White boxes represent the functions, and behaviors that achieve them are black rounded boxes. Within each function box is its name and set of provides conditions. Arrows are drawn from functions to behaviors to illustrate that association. Arrows from behaviors to functions represent the function-subfunction relationship with the origin extending from the superbehavior (and thus superfunction) and head of the arrow pointing to the subfunction. In this case, `PatchResistsPullout` is the topmost superfunction.

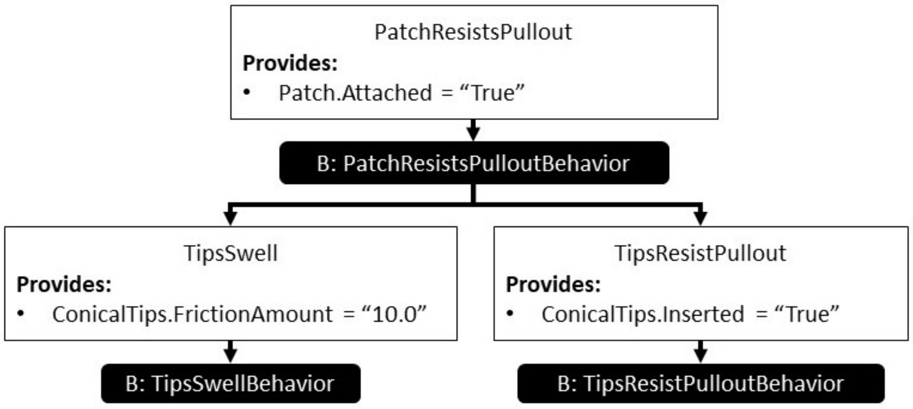

PatchResistsPullout
Provides:
• Patch.Attached = "True"
B: PatchResistsPulloutBehavior
TipsSwell
Provides:
• ConicalTips.FrictionAmount = "10.0"
B: TipsSwellBehavior
TipsResistPullout
Provides:
• ConicalTips.Inserted = "True"
B: TipsResistPulloutBehavior

Figure D.4: Functional decomposition of the Medical Patch model with mediating behaviors

D.3.2 The Model

```
Original_MedicalPatch is an SBF model

 The model has a structure

  PolystyreneCore is a component
   It has a connecting point named C1

  ConicalTips is a component
   It has an attribute named FrictionAmount of type
    Quantitative
   It has an attribute named Inserted of type Qualitative ,
    described as "False , True"
   It has an attribute named Size of type Qualitative ,
    described as "Small , Medium , Large , Maximum"
   It has an attribute named PullForceApplied of type
    Quantitative
   It has a connecting point named C1

  Polystyrene is a component
   It has a connecting point named C1

  PolyacrylicAcid is a component
   It has a connecting point named C1

  MicroneedleArray is a component
   It has an attribute named Applied of type Qualitative ,
    described as "False , True"
   It has an attribute named PullForceApplied of type
```

```
    Quantitative
   It has a connecting point named C1

  Patch is a component
   It has an attribute named Attached of type Qualitative,
    described as "False, True"
   It has an attribute named PullForceApplied of type
    Quantitative
   It has a connecting point named C1

  ConicalTips.C1 is OuterLayerOf PolystyreneCore.C1 is a
   connection named Connection1

  Polystyrene.C1 is PartOf ConicalTips.C1 is a connection
   named Connection2

  PolyacrylicAcid.C1 is PartOf ConicalTips.C1 is a connection
   named Connection3

  PolystyreneCore.C1 is PartOf MicroneedleArray.C1 is a
   connection named Connection4

  ConicalTips.C1 is PartOf MicroneedleArray.C1 is a connection
   named Connection5

  MicroneedleArray.C1 is In Patch.C1 is a connection named
   Connection6

 The model has functions

  PatchResistsPullout is a function
   It is denoted by the verb "Resist"
   It provides: (
    Patch.Attached = "True"
   )
   It is achieved by the behavior PatchResistsPulloutBehavior

  TipsResistPullout is a function
   It is denoted by the verb "Resist"
   It provides: (
    ConicalTips.Inserted = "True"
   )
   It is achieved by the behavior TipsResistPulloutBehavior

  TipsSwell is a function
   It is denoted by the verb "Grow"
   It provides: (
    ConicalTips.FrictionAmount = "10.0"
   )
   It is achieved by the behavior TipsSwellBehavior

 The model has behaviors

  PatchResistsPulloutBehavior is a behavior

   State1 is the start state
    It has the condition: (
     Patch.Attached = "False"
     and ConicalTips.Inserted = "False"
     and ConicalTips.FrictionAmount = "0.0"
     and MicroneedleArray.Applied = "False"
     and ConicalTips.Size = "Small"
     and Patch.PullForceApplied = "0.0"
    )

   T1 is a transition from State1 to State2
    It has the explanation: (
    the external stimulus PersonInsertsPatch, described as
     "eq: ConicalTips.Inserted is equal to
     ConicalTips.Inserted.True"
    and the connection Connection5, described as "eq:
     MicroneedleArray.Applied is directly proportional to
     the change in ConicalTips.Inserted"
    and the connection Connection6, described as "eq:
     Patch.Attached is directly proportional to the change
     in MicroneedleArray.Applied"
   )

  State2 is a state
   It has the condition: (
    Patch.Attached = "True"
    and ConicalTips.Inserted = "True"
    and MicroneedleArray.Applied = "True"
   )

  T2 is a transition from State2 to State3
   It has the explanation: (
    the function TipsSwell
   )

  State3 is a state
   It has the condition: (
    ConicalTips.FrictionAmount = "10.0"
    and ConicalTips.Size = "Large"
   )

  T3 is a transition from State3 to State4
   It has the explanation: (
    the external stimulus PullForceAppliedToPatch, described
     as "eq: Patch.PullForceApplied = 5.0"
   )

  State4 is a state
   It has the condition: (
    Patch.PullForceApplied = "5.0"
   )

  T4 is a transition from State4 to State5
   It has the explanation: (
    the function TipsResistPullout
   )

  State5 is the stop state
   It has the condition: (
    Patch.Attached = "True"
   )

 TipsSwellBehavior is a behavior

  State1 is the start state
   It has the condition: (
    ConicalTips.Size = "Small"
    and ConicalTips.Inserted = "True"
    and ConicalTips.FrictionAmount = "0.0"
   )

  T1 is a transition from State1 to State2
   It has the explanation: (
    the equation TipsStartSwelling, described as "eq:
     ConicalTips.Size is directly proportional to 1.0"
    and the equation LargerTipsMeansMoreResistanceForce,
     described as "eq: ConicalTips.FrictionAmount =
     ConicalTips.FrictionAmount:Before + 5.0"
    )

   State2 is a state
    It has the condition: (
     ConicalTips.Size = "Medium"
     and ConicalTips.FrictionAmount = "5.0"
    )

   T2 is a transition from State2 to State3
    It has the explanation: (
     the equation TipsFinishSwelling, described as "eq:
      ConicalTips.Size is directly proportional to 1.0"
     and the equation LargerTipsMeansMoreResistanceForce,
      described as "eq: ConicalTips.FrictionAmount =
      ConicalTips.FrictionAmount:Before + 5.0"
    )

   State3 is the stop state
    It has the condition: (
     ConicalTips.Size = "Large"
     and ConicalTips.FrictionAmount = "10.0"
    )

  TipsResistPulloutBehavior is a behavior

   State1 is the start state
    It has the condition: (
     Patch.Attached = "True"
     and ConicalTips.FrictionAmount = "10.0"
     and ConicalTips.Inserted = "True"
     and Patch.PullForceApplied = "5.0"
     and ConicalTips.PullForceApplied = "0.0"
     and MicroneedleArray.PullForceApplied = "0.0"
     and MicroneedleArray.Applied = "True"
    )

   T1 is a transition from State1 to State2
    It has the explanation: (
     the connection Connection6, described as "eq:
      MicroneedleArray.PullForceApplied =
      Patch.PullForceApplied:Before"
     and the connection Connection5, described as "eq:
      ConicalTips.PullForceApplied =
      MicroneedleArray.PullForceApplied:After"
    )

   State2 is a state
    It has the condition: (
     MicroneedleArray.PullForceApplied = "5.0"
     and ConicalTips.PullForceApplied = "5.0"
    )

   T2 is a transition from State2 to State3
    It has the explanation: (
     the equation ConicalTipsEffectedByCompetingForces,
      described as "eq: ConicalTips.Inserted is directly
      proportional to ConicalTips.FrictionAmount:Before -
      ConicalTips.PullForceApplied:Before"
     and the connection Connection5, described as "eq:
      MicroneedleArray.Applied is directly proportional to
      the change in ConicalTips.Inserted"
     and the connection Connection6, described as "eq:
      Patch.Attached is directly proportional to the change
      in MicroneedleArray.Applied"
    )
    State3 is the stop state
     It has the condition: (
      ConicalTips.Inserted = "True"
      and MicroneedleArray.Applied = "True"
      and Patch.Attached = "True"
     )
```

D.4 Owl Wing

D.4.1 Description and Functional Decomposition

This model was derived from two sources [74, 100]. This model describes how fimbriae on the owl's wing (which combine to form serrations on the wings) enable it to fly quietly. (Admittedly, the name of the function `OwlFliesSilently` in this model is a bit of a misnomer in retrospect since very low noise is created in this model, not zero.) Specifically, the model describes the owl deciding to fly, causing the air to move past it. The owl's fimbriae create micro-sized turbulences in the air, which in turn create a very low amount of noise.

Figure D.5 diagrammatically presents the functional decomposition of the Owl Wing model. White boxes represent the functions, and behaviors that achieve them are black rounded boxes. Within each function box is its name and set of provides conditions. Arrows are drawn from functions to behaviors to illustrate that association. Arrows from behaviors to functions represent the function-subfunction relationship with the origin extending from the superbehavior (and thus superfunction) and head of the arrow pointing to the subfunction. In this case, `OwlFliesSilently` is the topmost superfunction.

D.4.2 The Model

```
Original_OwlWing is an SBF model

 The model has a structure

  Owl is a component
   It has a connecting point named C1
   It has an attribute Moving of type Qualitative,
```

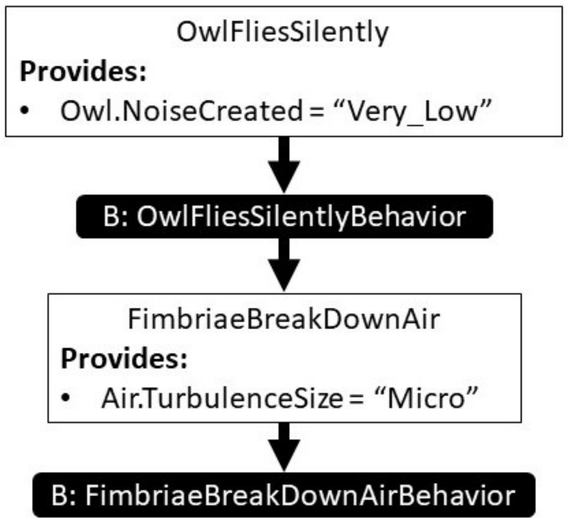

OwlFliesSilently
Provides:
Owl.NoiseCreated = “Very_Low”
B: OwlFliesSilentlyBehavior
FimbriaeBreakDownAir
Provides:
Air.TurbulenceSize = “Micro”
B: FimbriaeBreakDownAirBehavior

Figure D.5: Functional decomposition of the Owl Wing model with mediating behaviors

```
      described as "False, True"
     It has an attribute NoiseCreated of type Qualitative,
      described as "Zero, Very_Low, Low, Medium, High"

    Wing is a component
     It has a connecting point named C1

    Serrations is a component
     It has a connecting point named C1

    Air is a component
     It has a connecting point named C1
     It has an attribute MovingPastOwl of type Qualitative,
      described as "False, True"
     It has an attribute TurbulenceSize of type Qualitative,
      described as "Zero, Micro, Small, Medium, Large"

    Wing.C1 is PartOf Owl.C1 is a connection named
     WingIsPartOfAnOwl

    Serrations.C1 is PartOf Wing.C1 is a connection named
     OwlWingHasSerrations

   The model has functions

    OwlFliesSilently is a function
     It requires nothing
     It provides: (
      Owl.NoiseCreated = "Very_Low"
     )
     It is achieved by the behavior OwlFliesSilentlyBehavior

    FimbriaeBreakDownAir is a function
     It requires nothing
     It provides: (
      Air.TurbulenceSize = "Micro"
     )
     It is achieved by the behavior FimbriaeBreakDownAirBehavior
```

```
The model has behaviors
 OwlFliesSilentlyBehavior is a behavior

  StartState is the start state
   It has the condition: (
    Owl.Moving = "False"
    and Air.MovingPastOwl = "False"
    and Owl.NoiseCreated = "Zero"
   )

  T1 is a transition from StartState to State2
   It has the explanation: (
    the external stimulus named OwlDecidesToFly, described
     as "eq: Owl.Moving is equal to Owl.Moving.True"
    and the equation AirMovesPastOwlEQ, described as "eq:
     Air.MovingPastOwl is directly proportional to
     Owl.Moving:After - Owl.Moving:Before"
   )

  State2 is a state
   It has the condition: (
    Owl.Moving = "True"
    and Air.MovingPastOwl = "True"
   )

  T2 is a transition from State2 to StopState
   It has the explanation: (
    the function FimbriaeBreakDownAir
    and the equation TurbulenceCausesNoise, described as
     "eq: Owl.NoiseCreated is equal to Air.TurbulenceSize:After"
   )

  StopState is the stop state
   It has the condition: (
    Owl.NoiseCreated = "Very_Low"
   )

 FimbriaeBreakDownAirBehavior is a behavior

  StartState is the start state
   It has the condition: (
    Air.MovingPastOwl = "True"
    and Air.TurbulenceSize = "Zero"
   )

  T1 is a transition from StartState to StopState
   It has the explanation: (
    the connection OwlWingHasSerrations, described as "eq:
     Air.TurbulenceSize is directly proportional to
     Air.TurbulenceSize.Micro - Air.TurbulenceSize:Before"
   )

  StopState is the stop state
   It has the condition: (
    Air.TurbulenceSize = "Micro"
   )
```

D.5 Synthetic Car

D.5.1 Description and Functional Decomposition

This model was created as a synthetic training model for the Usefulness Study. As such, it has no source materials. This model describes a car transporting a passenger some distance and then stopping. Specifically, the model describes the driver turning on the engine, which then turns the wheels (adds to their RPM). The turning wheels add velocity to the car. After traveling some distance, the driver applies the brakes, stopping the wheels from turning and thus stopping the car.

Figure D.6 diagrammatically presents the functional decomposition of the Synthetic Car model. White boxes represent the functions, and behaviors that achieve them are black rounded boxes. Within each function box is its name and set of provides conditions. Arrows are drawn from functions to behaviors to illustrate that association. Arrows from behaviors to functions represent the function-subfunction relationship with the origin extending from the superbehavior (and thus superfunction) and head of the arrow pointing to the subfunction. In this case, `Car_Transports_a_Passenger` is the topmost superfunction.

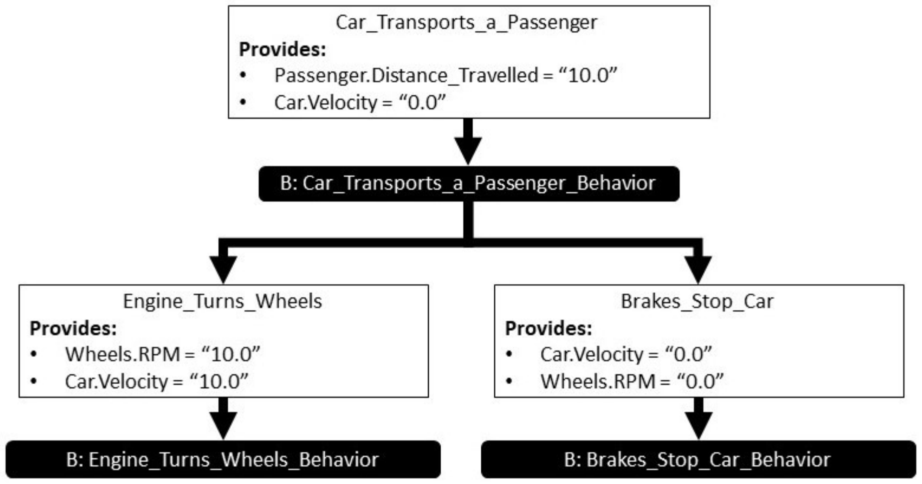


Figure D.6: Functional decomposition of the Synthetic Car model with mediating behaviors

D.5.2 The Model

```
Original_TrainingModel is an SBF model

 The model has a structure

  Passenger is a component
   It has a connecting point named C1
   It has an attribute Distance_Travelled of type Quantitative

  Car is a component
   It has a connecting point named C1
   It has an attribute Velocity of type Quantitative

  Wheels is a component
   It has a connecting point named C1
   It has an attribute RPM of type Quantitative

  Engine is a component
   It has a connecting point named C1
   It has an attribute State of type Descriptive

  Brakes is a component
   It has a connecting point named C1
   It has an attribute Engaged of type Descriptive
   It has an attribute RPM_Reduction_Amount of type
    Quantitative

 The model has behaviors

  Car_Transports_a_Passenger_Behavior is a behavior

   Start_State is the start state
    It has the condition: (
     Passenger.Distance_Travelled = "0.0"
     and Car.Velocity = "0.0"
     and Wheels.RPM = "0.0"
     and Engine.State = "Off"
     and Brakes.Engaged = "False"
    )

   State-2 is a state
    It has the condition: (
     Engine.State = "On"
    )

   State-3 is a state
    It has the condition: (
     Wheels.RPM = "10.0"
     and Car.Velocity = "10.0"
    )

   State-4 is a state
    It has the condition: (
     Passenger.Distance_Travelled = "10.0"
    )

   State-5 is a state
    It has the condition: (
     Brakes.Engaged = "True"
    )

   Stop_State is the stop state
    It has the condition: (
     Car.Velocity = "0.0"
     and Wheels.RPM = "0.0"
    )

   Transition-1 is a transition from Start_State to State-2
    It has the explanation (
     the stimulus Driver_Turns_on_Engine, described as "eq:
      Engine.State = On"
    )

   Transition-2 is a transition from State-2 to State-3
    It has the explanation (
     the function Engine_Turns_Wheels
    )

   Transition-3 is a transition from State-3 to State-4
    It has the explanation (
     the equation EQ_2, described as "eq:
      Passenger.Distance_Travelled = Passenger.Distance
      _Travelled:Before + Car.Velocity:Before"
    )

   Transition-4 is a transition from State-4 to State-5
    It has the explanation (
     the stimulus Driver_Engages_Brakes, described as "eq:
      Brakes.Engaged = True"
    )

   Transition-5 is a transition from State-5 to Stop_State
    It has the explanation (
     the function Brakes_Stop_Car
    )

  Engine_Turns_Wheels_Behavior is a behavior

   Start_State is the start state
    It has the condition: (
     Engine.State = "On"
     and Wheels.RPM = "0.0"
     and Car.Velocity = "0.0"
    )

   State_2 is a state
    It has the condition: (
     Wheels.RPM = "10.0"
    )

   Stop_State is the stop state
    It has the condition: (
     Car.Velocity = "10.0"
    )

   Transition-1 is a transition from Start_State to State_2
    It has the explanation (
     the equation EQ_1, described as "eq: Wheels.RPM =
      Wheels.RPM:Before + 10.0"
    )

   Transition-2 is a transition from State_2 to Stop_State
    It has the explanation (
     the principle Traction_of_the_tires_against_the_street,
      described as "eq: Car.Velocity = Car.Velocity:Before +
      Wheels.RPM:Before"
    )

  Brakes_Stop_Car_Behavior is a behavior
```

```
    State-1 is the start state
     It has the condition: (
      Brakes.Engaged = "True"
      and Car.Velocity = "10.0"
      and Brakes.RPM_Reduction_Amount = "10.0"
      and Wheels.RPM = "10.0"
     )

    State-2 is a state
     It has the condition: (
      Wheels.RPM = "0.0"
     )

    State-3 is the stop state
     It has the condition: (
      Car.Velocity = "0.0"
     )

    Transition-1 is a transition from State-1 to State-2
     It has the explanation (
      the equation Explanation-20, described as "eq: Wheels.RPM
       = Wheels.RPM:Before - Brakes.RPM_Reduction_Amount:Before"
     )

    Transition-2 is a transition from State-2 to State-3
     It has the explanation (
      the equation Velocity_proportional_to_wheel_RPMs,
       described as "eq: Car.Velocity = Wheels.RPM:Before"
     )

  The model has functions

   Car_Transports_a_Passenger is a function
    It requires nothing
    It is achieved by the behavior
     Car_Transports_a_Passenger_Behavior
    It provides: (
     Passenger.Distance_Travelled = "10.0"
     and Car.Velocity = "0.0"
    )

   Engine_Turns_Wheels is a function
    It requires nothing
    It is achieved by the behavior Engine_Turns_Wheels_Behavior
    It provides: (
     Wheels.RPM = "10.0"
     and Car.Velocity = "10.0"
    )

   Brakes_Stop_Car is a function
    It requires nothing
    It is achieved by the behavior Brakes_Stop_Car_Behavior
    It provides: (
     Car.Velocity = "0.0"
     and Wheels.RPM = "0.0"
    )
```

D.6 *Train*

D.6.1 Description and Functional Decomposition

This model was derived from the same two sources as the Owl Wing model [74, 100]. The model represents a train with a small vortex generator installed on it, inspired by how an owl reduces the amount of noise it makes while in flight. This was done to reduce the amount of aerodynamic noise that the train makes. The model describes how a train conductor puts the train into motion, leading to air flowing over one of the train's pantographs, which are physical structures extending out of the top of the train. Turbulence forms because of this air flow, but the vortex size moves to small because of the small vortex generator. These small vortices in turn cause the amount of noise generated by the train to move to low.

Note that this model actually contains an error. The `TurbulenceVortexSize` attribute of `Air` has a quantity with a space in it (`Very Large`), which is not legal. However, this specific situation should not have impacted the Comparison technique's results based on reflecting on how Comparison works. To ensure it did not effect Simulation, Simulation trials for the Train model were reran with a with a fixed version, and the simulator's outcomes were the same. It did produce slightly longer run times in all trials where run time was recorded, but these differences could simply be noise. Only the original run times were analyzed in the Computational Experimentation in Chapter 5.

Figure D.7 diagrammatically presents the functional decomposition of the Train model. White boxes represent the functions, and behaviors that achieve them are black rounded boxes. Within each function box is its name and set of provides conditions. Arrows are drawn from functions to behaviors to illustrate that association. Arrows from behaviors to functions represent the function-subfunction relationship with the origin extending from the superbehavior (and thus superfunction) and head of the arrow pointing to the subfunction. In this case, `TrainGeneratesAerodynamicNoise` is the topmost superfunction.

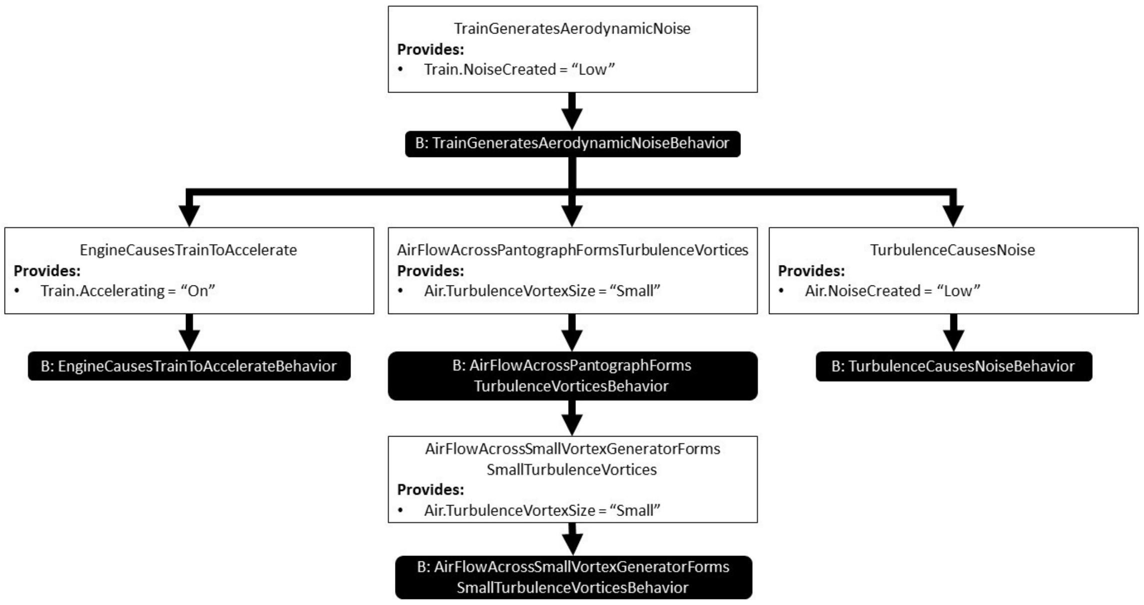

TrainGeneratesAerodynamicNoise
Provides:
• Train.NoiseCreated = "Low"
B: TrainGeneratesAerodynamicNoiseBehavior
EngineCausesTrainToAccelerate
Provides:
• Train.Accelerating = "On"
AirFlowAcrossPantographFormsTurbulenceVortices
Provides:
• Air.TurbulenceVortexSize = "Small"
TurbulenceCausesNoise
Provides:
• Air.NoiseCreated = "Low"
B: EngineCausesTrainToAccelerateBehavior
B: AirFlowAcrossPantographForms TurbulenceVorticesBehavior
B: TurbulenceCausesNoiseBehavior
AirFlowAcrossSmallVortexGeneratorForms SmallTurbulenceVortices
Provides:
• Air.TurbulenceVortexSize = "Small"
B: AirFlowAcrossSmallVortexGeneratorForms SmallTurbulenceVorticesBehavior

Figure D.7: Functional decomposition of the Train model with mediating behaviors

D.6.2 The Model

```
Original_TrainModel is an SBF model

 The model has a structure

  Train is a component
   It has a connecting point named C1
   It has an attribute Accelerating of type Qualitative,
    described as "Off, On"
   It has an attribute Velocity of type Qualitative,
    described as "Zero, Low, Medium, High, Maximum"
   It has an attribute NoiseCreated of type Qualitative,
    described as "Zero, Low, Medium, High, Maximum"
   It has an attribute EngineThrottle of type Qualitative,
    described as "Off, On"

  Pantograph is a component
   It has a connecting point named C1
   It has an attribute Velocity of type Qualitative,
    described as "Zero, Low, Medium, High, Maximum"

  Air is a component
   It has a connecting point named C1
   It has an attribute FlowOverPantograph of type Qualitative,
    described as "Zero, Low, Medium, High, Maximum"
   It has an attribute NoiseCreated of type Qualitative,
    described as "Zero, Low, Medium, High, Maximum"
   It has an attribute TurbulenceVortexSize of type
    Qualitative, described as "None, Small, Medium, Large,
    Very Large"
   It has an attribute FlowDirection of type Descriptive
  SmallVortexGenerator is a component
   It has a connecting point named C1
   It has an attribute VortexSizeProduced of type Qualitative ,
    described as "Zero , Low, Medium, High, Maximum"

  SmallVortexGenerator.C1 On Pantograph.C1 is a connection
   named SmallVortexGeneratorOnPantograph

  Pantograph.C1 is PartOf Train.C1 is a connection named
   PantographIsPartOfTrain

The model has functions

  TrainGeneratesAerodynamicNoise is a function
   It requires nothing
   It provides: (
    Train.NoiseCreated = "Low"
   )
   It is achieved by the behavior
    TrainGeneratesAerodynamicNoiseBehavior

  EngineCausesTrainToAccelerate is a function
   It requires nothing
   It provides: (
    Train.Accelerating = "On"
   )
   It is achieved by the behavior
    EngineCausesTrainToAccelerateBehavior

  AirFlowAcrossPantographFormsTurbulenceVortices is a function
   It requires nothing
   It provides: (
    Air.TurbulenceVortexSize = "Small"
   )
   It is achieved by the behavior
    AirFlowAcrossPantographFormsTurbulenceVorticesBehavior

  AirFlowAcrossSmallVortexGeneratorFormsSmallTurbulenceVortices
   is a function
   It requires nothing
   It provides: (
    Air.TurbulenceVortexSize = "Small"
   )
   It is achieved by the behavior AirFlowAcrossSmallVortex
    GeneratorFormsSmallTurbulenceVorticesBehavior

  TurbulenceCausesNoise is a function
   It requires nothing
   It provides: (
    Air.NoiseCreated = "Low"
   )
   It is achieved by the behavior
    TurbulenceCausesNoiseBehavior

The model has behaviors

  TrainGeneratesAerodynamicNoiseBehavior is a behavior

   StartState is the start state
    It has the condition (
     Train.Accelerating = "Off"
     and Train.Velocity = "Zero"
     and Train.NoiseCreated = "Zero"
     and Pantograph.Velocity = "Zero"
     and Air.FlowOverPantograph = "Zero"
    and Air.NoiseCreated = "Zero"
   )

 T1 is a transition from StartState to TrainAccelerates
  It has the explanation: (
   the function EngineCausesTrainToAccelerate
   and the equation E1, described as "eq: Pantograph.Velocity
    is directly proportional to the change in Train.Velocity"
   and the equation E2, described as "eq:
    Air.FlowOverPantograph is directly proportional to the
    change in Pantograph.Velocity"
  )

 TrainAccelerates is a state
  It has the condition: (
   Train.Accelerating = "On"
   and Train.Velocity = "Low"
   and Pantograph.Velocity = "Low"
   and Air.FlowOverPantograph = "Low"
  )

 T2 is a transition from TrainAccelerates to
  TrainContinuesAccelerating
  It has the explanation: (
   the equation E1, described as "eq: Train.Velocity is
    directly proportional to Train.Accelerating:After"
   and the equation E2, described as "eq: Pantograph.Velocity
    is directly proportional to the change in Train.Velocity"
   and the equation E3, described as "eq:
    Air.FlowOverPantograph is directly proportional to the
    change in Pantograph.Velocity"
  )

 TrainContinuesAccelerating is a state
  It has the condition: (
   Train.Velocity = "Medium"
   and Pantograph.Velocity = "Medium"
   and Air.FlowOverPantograph = "Medium"
  )

 T3 is a transition from TrainContinuesAccelerating to
  TrainReachesHighVelocity
  It has the explanation: (
   the equation E1, described as "eq: Train.Velocity is
    directly proportional to Train.Accelerating:After"
   and the equation E2, described as "eq: Pantograph.Velocity
    is directly proportional to the change in Train.Velocity"
   and the equation E3, described as "eq:
    Air.FlowOverPantograph is directly proportional to the
    change in Pantograph.Velocity"
  )

 TrainReachesHighVelocity is a state
  It has the condition: (
   Train.Velocity = "High"
   and Pantograph.Velocity = "High"
   and Air.FlowOverPantograph = "High"
  )

 T4 is a transition from TrainReachesHighVelocity to
  FlowingAirFormsTurbulence
  It has the explanation: (
   the function AirFlowAcrossPantographFormsTurbulence
    Vortices
  )
```

```
    FlowingAirFormsTurbulence is a state
     It has the condition: (
      Air.TurbulenceVortexSize = "Small"
     )

    T5 is a transition from FlowingAirFormsTurbulence to
     TurbulenceCausesNoise
     It has the explanation: (
      the function TurbulenceCausesNoise
      and the equation E1, described as "eq: Train.NoiseCreated
       is equal to Air.NoiseCreated:After"
     )

    TurbulenceCausesNoise is the stop state
     It has the condition: (
      Train.NoiseCreated = "Low"
     )

   EngineCausesTrainToAccelerateBehavior is a behavior

    TrainIsStill is the start state
     It has the condition: (
      Train.EngineThrottle = "Off"
      and Train.Accelerating = "Off"
      and Train.Velocity = "Zero"
     )

    T1 is a transition from TrainIsStill to TrainThrottleIsOn
     It has the explanation: (
      the external stimulus TrainConductorTurnsThrottleOn,
       described as "eq: Train.EngineThrottle is equal to
       Train.EngineThrottle.On"
     )

    TrainThrottleIsOn is a state
     It has the condition: (
      Train.EngineThrottle = "On"
     )

    T2 is a transition from TrainThrottleIsOn to
     TrainIsAccelerating
     It has the explanation: (
      the equation E1, described as "eq: Train.Accelerating is
       directly proportional to Train.EngineThrottle:Before"
      and the equation E2, described as "eq: Train.Velocity is
       directly proportional to Train.Accelerating:After"
     )

    TrainIsAccelerating is the stop state
     It has the condition: (
      Train.Velocity = "Low"
      and Train.Accelerating = "On"
     )

   AirFlowAcrossPantographFormsTurbulenceVorticesBehavior is a
    behavior

    StartState is the start state
     It has the condition: (
     Air.FlowDirection = "Towards Pantograph"
     and Air.TurbulenceVortexSize = "None"
    )

    T1 is a transition from StartState to
     AirFlowOnSmallVortexGenerator
     It has the explanation: (
      the connection SmallVortexGeneratorOnPantograph,
       described as "eq: Air.FlowDirection = Towards Small Vortex
       Generator"
     )

    AirFlowOnSmallVortexGenerator is a state
     It has the condition: (
      Air.FlowDirection = "Towards Small Vortex Generator"
     )

    T2 is a transition from AirFlowOnSmallVortexGenerator to
     StopState
     It has the explanation: (
      the function named AirFlowAcrossSmallVortexGeneratorForms
       SmallTurbulenceVortices
     )

    StopState is the stop state
     It has the condition: (
      Air.TurbulenceVortexSize = "Small"
     )

   AirFlowAcrossSmallVortexGeneratorFormsSmallTurbulence
    VorticesBehavior is a behavior

    StartState is the start state
     It has the condition: (
      Air.FlowDirection = "Towards Small Vortex Generator"
      and Air.TurbulenceVortexSize = "None"
      and SmallVortexGenerator.VortexSizeProduced = "Low"
     )

    T1 is a transition from StartState to StopState
     It has the explanation: (
      the equation E1, described as "eq:
       Air.TurbulenceVortexSize is directly proportional to
       SmallVortexGenerator.VortexSizeProduced:Before -
       Air.TurbulenceVortexSize:Before"
     )

    StopState is the stop state
     It has the condition: (
      Air.TurbulenceVortexSize = "Small"
     )

   TurbulenceCausesNoiseBehavior is a behavior

    StartState is the start state
     It has the condition: (
      Air.TurbulenceVortexSize = "Small"
      and Air.NoiseCreated = "Zero"
     )

    T1 is a transition from StartState to
     NoiseIncreasingFromTurbulence
     It has the explanation: (
      the equation E1, described as "eq: Air.NoiseCreated is
       directly proportional to Air.TurbulenceVortexSize:Before
      - Air.NoiseCreated:Before"
     )

    NoiseIncreasingFromTurbulence is the stop state
     It has the condition: (
      Air.NoiseCreated = "Low"
     )
```

REFERENCES

[1] Ahmed, S., Wallace, K. M., and Blessing, L. T., "Understanding the differences between how novice and experienced designers approach design tasks," *Research in Engineering Design*, vol. 14, pp. 1–11, Feb. 2003.

[2] Arciszewski, T., Bloedorn, E., Michalski, R. S., Mustafa, M., and Wnek, J., "Machine learning of design rules: methodology and case study," *Journal of Computing in Civil Engineering*, vol. 8, July 1994.

[3] Arciszewski, T. and Cornell, J., "Bio-inspiration: Learning creative design principia," *Intelligent Computing in Engineering and Architecture, Lecture Notes in Computer Science*, vol. 4200, pp. 32–53, 2006.

[4] Athavankar, U. A., "Mental imagery as a design tool," *Cybernetics and Systems*, vol. 28, no. 1, pp. 25–42, 1997.

[5] Ball, L. J. and Christensen, B. T., "Analogical reasoning and mental simulation in design: two strategies linked to uncertainty resolution," *Design Studies*, vol. 30, pp. 169–186, Mar. 2009.

[6] Balzer, R., "A 15 year perspective on automatic programming," *IEEE Transactions on Software Engineering*, vol. SE-11, pp. 1257–1268, Nov. 1985.

[7] Bar-Cohen, Y., ed., *Biomimetics: Biologically Inspired Technologies*. Taylor & Francis, 2005.

[8] Barber, J., Bhatta, S., Goel, A., Jacobson, M., Pearce, M., Penberthy, L., Shankar, M., Simpson, R., and Stroulia, E., "AskJef: Integration of case-based and multimedia technologies for interface design support," *Artificial Intelligence in Design '92*, pp. 457–475, 1992.

[9] Benyus, J., *Biomimicry: Innovation Inspired by Nature*. William Morrow, 1997.

[10] Bhatta, S. and Goel, A., "Model-based design indexing and index learning in engineering design," *Engineering Applications of Artificial Intelligence*, vol. 9, no. 6, pp. 601–609, 1996.

[11] Bhushan, B., "Biomimetics: lessons from nature–an overview," *Philosophical Transactions of the Royal Society A*, vol. 367, no. 1893, 2009.

[12] Boehm, B. W., "Verifying and validating software requirements and design specifications," *IEEE Software*, vol. 1, no. 1, pp. 75–88, 1984.

[13] Bonnardel, N., "Towards understanding and supporting creativity in design: analogies in a constrained cognitive environment," *Knowledge-Based Systems*, vol. 13, pp. 505–513, Dec. 2000.

[14] Bonnardel, N. and Sumner, T., "Supporting evaluation in design," *Acta Psychologica*, pp. 221–244, 1996.

[15] Bonser, R. and Vincent, J., "Technology trajectories, innovation, and the growth of biomimetics," *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, pp. 1177–1180, 2007.

[16] Börner, K., Pippig, E., Tammer, E. C., and Coulon, C. H., *Advances in Case-Based Reasoning. EWCBR 1996. Lecture Notes in Computer Science (Lecture Notes in Artificial Intelligence)*, vol. 1168, ch. Structural similarity and adaptation, pp. 58–75. Berlin, Heidelberg: Springer, 1996.

[17] Brown, D. C. and Chandrasekaran, B., *Problem Solving: Knowledge Structures and Control Strategies*. Pitman Publishing, 1989.

[18] Bruck, H., Gershon, A., Golden, I., Gupta, S., Gyger, L., Magrab, E., and Spranklin, B., "Training mechanical engineering students to utilize biological inspiration during product development," *Bioinspiration and Biomimetics*, vol. 2, pp. S198–S209, 2007.

[19] Casakin, H. and Goldschmidt, G., "Expertise and the use of visual analogy: implications for design education," *Design Studies*, vol. 20, pp. 153–175, Mar. 1999.

[20] Chakrabarti, A., Sarkar, P., Leelavathama, B., and Nataraju, B., "A functional representation for aiding biomimetic and artificial inspiration of new ideas," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, vol. 19, no. 2, pp. 113–132, 2005.

[21] Chakrabarti, A., Siddharth, L., Dinakar, M., Panda, M., Palegar, N., and Keshwani, S., "Idea inspire 3.0–a tool for analogical design," in *Research into Design for Communities, Volume 2. ICoRD 2017. Smart Innovation, System and Technologies* (Chakrabarti, A. and Chakrabarti, D., eds.), vol. 66, Singapore: Springer, 2017.

[22] Chan, J., Fu, K., Schunn, C., Cagan, J., Wood, K., and Kotovsky, K., "On the benefits and pitfalls of analogies for innovative design: ideation performance based on analogical distance, commonness, and modality of examples," *Journal of Mechanical Design*, vol. 133, Aug. 2011.

[23] Chandrasekaran, B., "Design problem solving: A task analysis," *AI Magazine*, vol. 11, no. 4, pp. 59–71, 1990.

[24] Chandrasekaran, B., "Functional representation: a brief historical perspective," *Applied Artificial Intelligence*, vol. 8, no. 2, pp. 173–197, 1994.

[25] Chandrasekaran, B., "Functional representation and causal processes," *Advances in Computers*, vol. 38, pp. 73–143, 1994.

[26] Chandrasekaran, B., "Representing function: Relating functional representation and functional modeling research streams," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, vol. 19, pp. 65–74, May 2005.

[27] Chandrasekaran, B., Goel, A., and Iwasaki, Y., "Functional representation as design rationale," *IEEE Computer*, vol. 26, pp. 48–56, Jan. 1993.

[28] Cheong, H., Chiu, I., Shu, L. H., and Stone, R. B., "Biologically meaningful keywords for functional terms of the functional basis," *Journal of Mechanical Design*, vol. 133, no. 2, pp. 021007–1–021007–11, 2011.

[29] Chin, K.-s. and Wong, T. N., "Knowledge-based evaluation for the conceptual design development of injecting molding parts," *Engineering Applications of Artificial Intelligence*, vol. 9, no. 4, pp. 359–376, 1996.

[30] Chiu, I. and Shu, L., "Biomimetic design through natural language analysis to facilitate cross-domain information retrieval," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, vol. 21, no. 1, pp. 45–59, 2007.

[31] Christensen, B. T. and Schunn, C. D., "The relationship of analogical distance to analogical function and preinventive structure: the case of engineering design," *Memory & Cognition*, vol. 35, pp. 29–38, Jan. 2007.

[32] Cross, N., "The nature and nurture of design ability," *Design Studies*, vol. 11, no. 3, pp. 127–140, 1990.

[33] Cutkosky, M. R., Engelmore, R. S., Fikes, R. E., Genesereth, M. R., Gruber, T. R., Mark, W. S., Tenenbaum, J. M., and Weber, J. C., "PACT: an experiment in integrating concurent engineering systems," *Computer*, vol. 26, pp. 28–37, Jan. 1993.

[34] Dahl, D. W. and Moreau, P., "The influence and value of analogical thinking during new product ideation," *Journal of Marketing Research*, vol. 39, pp. 47–60, Feb. 2002.

[35] D'Amelio, V., Chmarra, M. K., and Tomiyama, T., "Early design interference detection based on qualitative physics," *Research in Engineering Design*, vol. 22, pp. 223–243, 2011.

[36] Davies, J., Goel, A. K., and Nersessian, N. J., "A computational model of visual analogies in design," *Cognitive Systems Research, Special Issue on Analogies - Integrating Cognitive Abilities*, vol. 10, pp. 204–215, Sept. 2009.

[37] de Gómez Silva Garza, A. and Maher, M. L., *Case-Based Reasoning Research and Development. ICCBR 1999. Lecture Notes in Computer Science (Lecture Notes in Artificial Intelligence)*, vol. 1650, ch. An Evolutionary Approach to Case Adaptation, pp. 162–173. Berlin, Heidelberg: Springer, 1999.

[38] Dean, B. and Bhushan, B., "Shark-skin surfaces for fluid-drag reduction in turbulent flow: a review," *Phil. Trans. R. Soc. A*, vol. 368, pp. 4775–4806, 2010.

[39] Deldin, J.-M. and Schuknecht, M., "The AskNature database: Enabling solutions in biomimetic design," in *Biologically Inspired Design* (Goel, A. K., McAdams, D., and Stone, R., eds.), pp. 12–27, London: Springer, 2014.

[40] Dym, C. and Brown, D., *Engineering design: Representation and reasoning*. New York: Cambridge University Press, 2nd ed., 2012.

[41] Eastman, C. M., Bond, A. H., and Chase, S. C., "A data model for design databases," *Artificial Intelligence in Design '91*, pp. 339–365, 1991.

[42] Eliasmith, C. and Thagard, P., "Integrating structure and meaning: a distributed model of analogical mapping," *Cognitive Science*, vol. 25, no. 2, pp. 245–286, 2001.

[43] Falkenhainer, B., "An examination of the third stage in the analogy process: Verification-based analogical learning," *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, vol. 1, no. 260-263, 1987.

[44] Falkenhainer, B., Forbus, K. D., and Gentner, D., "The structure-mapping engine: Algorithm and examples," *Artificial Intelligence*, vol. 41, pp. 1–63, 1989.

[45] Faltings, B. and Sun, K., "FAMING: supporting innovative mechanism shape design," *Computer-Aided Design*, vol. 28, no. 3, pp. 207–216, 1996.

[46] Finger, S. and Dixon, J. R., "A review of research in mechanical engineering design. part I: Descriptive, prescriptive, and computer-based models of design processes," *Research in Engineering Design*, vol. 1, pp. 51–67, Mar. 1989.

[47] Fischer, G., Grudin, J., Lemke, A., McCall, R., Ostwald, J., Reeves, B., and Shipman, F., "Supporting indirect collaborative design with integrated knowledge-based design environments," *Human-Computer Interaction*, vol. 7, no. 3: Computer-Supported Cooperative Work, pp. 281–314, 1992.

[48] Fischer, G., Nakakoji, K., Ostwald, J., Stahl, G., and Sumner, T., "Embedding computer-based critics in the contexts of design," *CHI '93 Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems*, pp. 157–164, Apr. 1993.

[49] Fish, F., "Imaginative solutions by marine organisms for drag reduction," *Proceedings of the International Symposium On Seawater Drag Reduction*, 1998. Newport, RI.

[50] Fish, F. E., Weber, P. W., Murray, M. M., and Howle, L. E., "The tubercles on humpback whales' flippers: Application of bio-inspired technology," *Integrative and Comparative Biology*, vol. 51, pp. 203–213, July 2011.

[51] Flemming, U., "Case-based design in the SEED system," *Automation in Construction*, vol. 3, pp. 123–133, July 1994.

[52] Forbus, K. D., Gentner, D., Everett, J. O., and Wu, M., "Towards a computational model of evaluating and using analogical inferences," *Proceedings of the 19th Annual Conference of the Cognitive Science Society*, pp. 229–234, 1997.

[53] Forbus, K. D., Gentner, D., and Law, K., "MAC/FAC: A model of similarity-based retrieval," *Cognitive Science*, vol. 19, no. 2, pp. 141–205, 1995. April-June.

[54] Gallagher, B., "Matching structure and semantics: a survey on graph-based pattern matching," *AAAI Fall Symposium on Capturing and Using Patterns for Evidence Detection*, 2006.

[55] Gebeshuber, I. C., Gruber, P., and Drack, M., "A gaze into the crystal ball: Biomimetics in the year 2059," *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, vol. 223, no. 12, pp. 2899–2918, 2009.

[56] Gebhardt, F., Voss, A., Gräther, W., and Schmidt-Belz, B., *Reasoning with Complex Cases*. Boston, MA: Springer, 1997.

[57] Gero, J. and Kannengiesser, U., "The situated function-behaviour-structure framework," *Design Studies*, vol. 25, no. 4, pp. 373–391, 2004.

[58] Goel, A., "Design, analogy, and creativity," *IEEE Expert*, vol. 12, no. 3, pp. 62–70, 1997. May-June.

[59] Goel, A., "A 30-year case study and 15 principles: Implications of an artificial intelligence methodology for functional modeling," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, vol. 27, no. 3, pp. 203–215, 2013.

[60] Goel, A., de Silver Garza, A. G., Grué, N., Murdock, J. W., Recker, M., and Govindaraj, T., "Explanatory interface in interactive design environments," in *Artificial Intelligence in Design '96* (Gero, J. S. and Sudweeks, F., eds.), pp. 387–405, Kluwer Academic Publishers, 1996.

[61] Goel, A., McAdams, D., and Stone, R., eds., *Biologically inspired design: Computational methods and tools*. London: Springer-Verlag, 2014.

[62] Goel, A. K. and Craw, S., "Design, innovation and case-based reasoning," *The Knowledge Engineering Review*, vol. 20, pp. 271–276, Sept. 2005.

[63] Goel, A. K. and Stroulia, E., "Functional device models and model-based diagnosis in adaptive design," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, vol. 10, pp. 355–370, Sept. 1996. Special Issue: Representing functionality in design.

[64] Goel, A. K. and Wiltgen, B., "On the role of analogy in resolving cognitive dissonance in collaborative interdisciplinary design," *Case-Based Reasoning Research and Development. ICCBR 2014. Lecture Notes in Computer Science*, vol. 8765, pp. 185–199, 2014. Lamontagne, L. & Plaza, E. (eds.). Springer, Cham.

[65] Goel, A., Rugaber, S., and Vattam, S., "Structure, behavior, and function of complex systems: The structure, behavior, and function modeling language," *International Journal of AI in Engineering Design, Analysis and Manufacturing: Special Issue on Developing and Using Engineering Ontologies*, vol. 23, pp. 23–35, Feb. 2009.

[66] Goel, A. K. and Bhatta, S. R., "Use of design patterns in analogy-based design," *Advanced Engineering Informatics*, vol. 18, pp. 85–94, 2004.

[67] Goel, A. K., Bhatta, S. R., and Stroulia, E., "KRITIK: An early case-based design system," in *Issues and Applications of Case-Based Reasoning in Design* (Maher, M. and Pu, P., eds.), pp. 87–132, Lawrence Erlbaum Associates, Inc., 1997.

[68] Gorti, S. R. and Sriram, R. D., "From symbol to form: a framework for conceptual design," *Computer-Aided Design*, vol. 28, pp. 853–870, Nov. 1996.

[69] Goswami, U. and Brown, A. L., "Melting chocolate and melting snowmen: analogical reasoning and causal relations," *Cognition*, vol. 35, pp. 69–95, Apr. 1990.

[70] Gruber, J., "Daring fireball: Markdown," 2018. Last retrieved from https://daringfireball.net/projects/markdown/ on July 22, 2018.

[71] Helms, M. and Goel, A., "The four-box method: Problem formulation and analogy evaluation in biologically inspired design," *Journal of Mechanical Design*, vol. 136, pp. 111106–1–111106–12, Oct. 2014.

[72] Helms, M., Vattam, S., and Goel, A., "Biologically inspired design: process and products," *Design Studies*, vol. 30, no. 5, pp. 606–622, 2009.

[73] Helms, M. E. and Goel, A. K., "Analogical problem evolution in biologically inspired design," in *Design Computing and Cognition '12* (Gero, J., ed.), pp. 3–19, Dordrecht: Springer, 2014.

[74] Hoeller, N., "Tools: Structure-behavior-function and functional modeling," *Zygote Quarterly*, vol. Spring 2013, pp. 150–167, 2013.

[75] Holyoak, K. J. and Thagard, P., *Mental Leaps: Analogy in Creative Thought*. Cambridge, Massachusetts: The MIT Press, 1995.

[76] Holyoak, K. J. and Thagard, P., "Analogical mapping by constraint satisfaction," *Cognitive Science*, vol. 13, no. 3, pp. 295–355, 1989. July-September.

[77] Hua, K., Fairings, B., and Smith, I., "CADRE: case-based geometric design," *Artificial Intelligence in Engineering*, vol. 10, no. 2, pp. 171–183, 1996.

[78] Iwasaki, Y., Vescovi, M., Fikes, R., and Chandrasekaran, B., "A causal functional representation language with behavior-based semantics," *Applied Artificial Intelligence*, vol. 9, pp. 5–31, 1995.

[79] Jansson, D. G. and Smith, S. M., "Design fixation," *Design Studies*, vol. 12, pp. 3–11, Jan. 1991.

[80] Joyner, D. A., Goel, A. K., and Papin, N. M., "MILA–S: generation of agent-based simulations from conceptual models of complex systems," *Proceedings of the 19th international conference on Intelligent User Interfaces*, pp. 289–298, Feb. 2014. Haifa, Israel.

[81] Katz, R. H., Chang, E., and Bhateja, R., "Version modeling concepts for computer-aided design databases," *In proceedings of the 1986 ACM SIGMOD international conference on management of data (SIGMOD '86)*, pp. 379–386, May 1986.

[82] Keane, M. T., Ledgeway, T., and Duff, S., "Constraints on analogical mapping: A comparison of three models," *Cognitive Science*, vol. 18, no. 3, pp. 387–438, 1994.

[83] Kitamura, Y., Kashiwase, M., Fuse, M., and Mizoguchi, R., "Deployment of an ontological framework for functional design knowledge," *Advanced Engineering Informatics*, vol. 18, no. 2, pp. 115–127, 2004.

[84] Kitamura, Y., Toshinobu, S., Namba, K., and Mizoguchi, R., "A functional concept ontology and its application to automatic identification of functional structures," *Advanced Engineering Informatics*, vol. 16, no. 2, pp. 145–163, 2002.

[85] Klenk, M., de Kleer, J., Bobrow, D., Yoon, S., Hanley, J., and Janssen, B., "Guiding and verifying early design using qualitative simulation," *Proceedings of the ASME 2012 International Design Engineering Technical Confrences & Computers and Information in Engineering Conference IDETC/CIE 2012*, 2012. Chicago, IL.

[86] Komoto, H. and Tomiyama, T., "A framework for computer-aided conceptual design and its application to system architecting of mechatronics products," *Computer-Aided Design*, vol. 44, pp. 931–946, 2012.

[87] Kruger, C. and Cross, N., "Solution driven versus problem driven design: strategies and outcomes," *Design Studies*, vol. 27, pp. 527–548, Sept. 2006.

[88] Lenau, T., "Biomimetics as a design methodology - possibilities and challenges," *International Conference on Engineering Design, ICED'09*, Aug. 2009. Stanford University, Stanford, CA, USA.

[89] Lindemann, U. and Gramann, J., "Engineering design using biological principles," *Proceedings of the 8th International Design Conference - DESIGN 2004*, pp. 355–360, 2004. Zagreb.

[90] Linsey, J. S., Wood, K. L., and Markman, A. B., "Modality and representation in analogy," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, vol. 22, pp. 85–100, 2008.

[91] Louis, S. J., "Working from blueprints: evolutionary learning in design," *Artificial Intelligence in Engineering*, vol. 11, pp. 335–341, July 1997.

[92] MacLellan, C., Langley, P., Shah, J., and Dinar, M., "A conceptual aid for problem formulation in early conceptual design," *ASME Journal of Computing and Information Science in Engineering*, vol. 13, no. 3, 2013.

[93] Maher, M. L., Balachandran, M. B., and Zhang, D. M., *Case-Based Reasoning in Design*. New York: Psychology Press, Aug. 1995.

[94] Maher, M. L. and de Silva Garza, A. G., "Case-based reasoning in design," *IEEE Expert*, vol. March-April, pp. 34–41, 1997.

[95] Maher, M. L. and Pu, P., eds., *Issues and Applications of Case-Based Reasoning to Design*. New York: Psychology Press, Apr. 1997.

[96] Mak, T. and Shu, L., "Using descriptions of biological phenomena for idea generation," *Research in Engineering Design*, vol. 19, no. 1, pp. 21–28, 2008.

[97] Marcus, S., Stout, J., and McDermott, J., "VT: an expert elevator designer that uses knowledge-based backtracking," *AI Magazine*, vol. 8, pp. 41–58, Dec. 1987. Winter.

[98] McConney, M. E., Chen, N., Lu, D., Hu, H. A., Coombs, S., Liu, C., and Tsukruk, V. V., "Biologically inspired design of hydrogel-capped hair sensors for enhanced underwater flow detection," *Soft Matter*, no. 2, 2009.

[99] McDonough, E. F., I., Kahnb, K. B., and Barczaka, G., "An investigation of the use of global, virtual, and colocated new product development teams," *Journal of Product Innovation Management*, vol. 18, pp. 110–120, Mar. 2001.

[100] McKeag, T., "Special feature: Auspicious forms," *Zygote Quarterly*, vol. summer 2012, pp. 14–33, 2012.

[101] McKeag, T., "Case study: Sticky wicket: A search for an optimal adhesive for surgery," *Zygote Quarterly*, vol. 1, 2015, pp. 18–41, 2015.

[102] Moss, J., Kotovsky, K., and Cagan, J., "The role of functionality in the mental representations of engineering students: some differences in the early stages of expertise," *Cognitive Science*, vol. 30, no. 1, pp. 65–93, 2006.

[103] Mostow, J., Barley, M., and Weinrich, T., "Automated reuse of design plans," *Artificial Intelligence in Engineering*, vol. 4, pp. 181–196, Oct. 1989.

[104] Murdock, J. W., Szykman, S., and Sriram, R. D., "An information modeling framework to support design databases and repositories," *In proceedings of the 1997 ASME Design Engineering Technical Conferences (DETC'97)*, Sept. 1997. Sacramento, California.

[105] Nagel, J. K. S., Nagel, R. L., Stone, R. B., and McAdams, D. A., "Function-based, biologically inspired concept generation," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, vol. 24, pp. 521–535, Nov. 2010.

[106] Nagel, J. K. S., Stone, R. B., and McAdams, D. A., "An engineering-to-biology thesaurus for engineering design," *ASME 2010 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference. 22nd International Conference on Design Theory and Methodology; Special Conference on Mechanical Vibration and Noise*, vol. 5, pp. 117–128, Aug. 2010. Montreal, Quebec, Canada.

[107] Nagel, R., Midha, P., Tinsley, A., Stone, R., McAdams, D., and Shu, L., "Exploring the use of functional models in biomimetic conceptual design," *ASME Journal of Mechanical Design*, vol. 130, no. 12, 2008.

[108] Nagel, R. L., Vucovich, J. P., Stone, R. B., and McAdams, D. A., "A signal grammar to guide functional modeling of electromechanical products," *Journal of Mechanical Design*, vol. 130, Mar. 2008.

[109] Nakrani, S. and Tovey, C., "On honey bees and dynamic server allocation in internet hosting centers," *Adaptive Behavior*, Dec. 2004.

[110] Oh, Y., Gross, M. D., Ishizaki, S., and Do, E. Y.-L., "Constraint-based design critic for flat-pack furniture design," *Proceedings of the 17th International Conference on Computers in Education [CDROM]*, 2009.

[111] Oxman, R., "Design by re-representation: a model of visual reasoning in design," *Design Studies*, vol. 18, pp. 329–347, Oct. 1997.

[112] Pahl, G., Beitz, W., Feldhusen, J., and Grote, K.-H., *Engineering Design: A Systematic Approach*. Springer, third ed., 2007.

[113] Pastor, O., Gómez, J., Insfrán, E., and Pelechano, V., "The OO-method approach for information systems modeling: from object-oriented conceptual modeling to automated programming," *Information Systems*, vol. 26, pp. 507–534, 2001.

[114] Pearce, M., Goel, A. K., Kolodner, J. L., Zimring, C., Sentosa, L., and Billington, R., "Case-based design support: A case study in architectural design," *IEEE Expert*, vol. 7, no. 5, pp. 14–20, 1992.

[115] Prabhakar, S. and Goel, A. K., "Functional modeling for enabling adaptive design of devices for new environments," *Artificial Intelligence in Engineering*, vol. 12, no. 4, pp. 417–444, 1998.

[116] Price, C., "Function-directed electrical design analysis," *Artificial Intelligence in Engineering*, vol. 12, pp. 445–456, 1998.

[117] Price, C., Travé-Massuyès, L., Milne, R., Ironi, L., Forbus, K., Bredeweg, B., Lee, M. H., Struss, P., Snooke, N., Lucas, P., Cavazza, M., and Coghill, G., "Qualitative futures," *Knowledge Engineering Review*, vol. 21, pp. 317–334, 2006.

[118] Qualitative Reasoning & Modeling, "Garp3 software," Last retrieved from https://ivi.fnwi.uva.nl/tcs/QRgroup/QRM/software/ on May 14, 2018.

[119] Qualitative Reasoning Group, "Case mapper," 2018. Last retrieved from http://www.qrg.northwestern.edu/software/casemapper/ on May 13, 2018. Version 7.04 (32-bit) for Windows was used in this dissertation.

[120] Rasmussen, J., "The role of hierarchical knowledge representation in decisionmaking and system management," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-15, no. 2, pp. 234–243, 1985. March-April.

[121] Richland, L. E., Holyoak, K. J., and Stigler, J. W., "Analogy use in eighth-grade mathematics classrooms," *Cognition and Instruction*, vol. 22, pp. 37–60, 2004.

[122] Robbins, J. E. and Redmiles, D. F., "Software architecture critics in the argo design environment," *Knowledge-Based Systems*, vol. 11, pp. 47–60, Sept. 1998.

[123] Ross, B. H. and Kilbane, M. C., "Effects of principle extraction and superficial similarity on analogical mapping in problem solving," *Journal of Experimental Psychology: Learning, Memory, and Cognition*, vol. 23, pp. 427–440, Mar. 1997.

[124] Rugaber, S., Bhati, S., Goswami, V., Spiliopoulou, E., Azad, S., Koushik, S., Kulkarni, R., Kumble, M., Sarathy, S., and Goel, A., "Knowledge extraction and annotation for cross-domain textual case-based reasoning in biologically inspired design," in *Case-Based Reasoning Research and Development. ICCBR 2016. Lecture Notes in Computer Science* (Goel, A., Díaz-Agudo, M., and Roth-Berghofer, T., eds.), vol. 9969, pp. 342–355, Cham: Springer, 2016.

[125] Rugaber, S., "Explanatarium," 2018. Last accessed from http://explanatarium.org on May 14, 2018.

[126] Runeson, P., "A survey of unit testing practices," *IEEE Software*, vol. July/August, pp. 22–29, 2006.

[127] Sartori, J., Pal, U., and Chakrabarti, A., "A methodology for supporting "transfer" in biomimetic design," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, vol. 24, pp. 483–506, Nov. 2010.

[128] Schuman, H. and Rieger, C., "Historical analogies, generational effects, and attitudes toward war," *American Sociological Review*, vol. 57, pp. 315–326, June 1992.

[129] Sembugamorthy, V. and Chandrasekaran, B., *Experience, Memory, and Reasoning*, ch. Functional representation of devices and compilation of diagnostic problem-solving systems, pp. 47–73. Mahwah, NJ: Lawrence Erlbaum Associates, Inc., 1986.

[130] Shanahan, M., "The frame problem," *The Stanford Encyclopedia of Philosophy (Spring 2016 Edition)*, 2016. Edward N. Zalta (ed.). Last retrieved on June 23, 2018 from https://plato.stanford.edu/archives/spr2016/entries/frame-problem/.

[131] Shasha, D., Wang, J. T. L., and Giugno, R., "Algorithmics and applications of tree and graph searching," *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pp. 39–52, June 2002.

[132] Shu, L. H., Stone, R. B., McAdams, D. A., and Greer, J. L., "Integrating function-based and biomimetic design for automatic concept generation," *In proceedings of International conference on engineering design, ICED'07*, Aug. 2007. 28 - 31 August. Cité des sciences et de l'industrie, Paris, France.

[133] Shu, L. H., Ueda, K., Chiu, I., and Cheong, H., "Biologically inspired design," *CIRP Annals*, vol. 60, no. 2, pp. 673–693, 2011.

[134] Silverman, B. G. and Mezher, T. M., "Expert critics in engineering design: Lessons learned and research needs," *AI Magazine*, vol. 13, no. 1, 1992. Spring.

[135] Sim, S. K. and Duffy, A. H. B., "A foundation for machine learning in design," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, vol. 12, pp. 193–209, Apr. 1998.

[136] Smith, I., Lottaz, C., and Faltings, B., *Case-Based Reasoning Research and Development. ICCBR 1995. Lecture Notes in Computer Science (Lecture Notes in Artificial Intelligence)*, vol. 1010, ch. Spatial composition using cases: IDIOM, pp. 88–97. Berlin, Heidelberg: Springer, 1995.

[137] Smyth, B. and Keane, M. T., *Case-Based Reasoning: Experiences, Lessons, & Future Directions*, ch. Design à la Déjà Vu: Reducing the Adaptation Overhead, pp. 151–166. Cambridge, Massachusetts: The AAAI Press/The MIT Press, 1996.

[138] Smyth, B., Keane, M. T., and Cunningham, P., "Hierarchical case-based reasoning integrating case-based and decompositional problem-solving techniques for plant-control software design," *IEEE Transactions on Knowledge and Data Engineering*, vol. 13, no. 5, pp. 793–812, 2001. September/October.

[139] Stahovich, T. F., "Chapter seven: Artificial intelligence for design," in *Formal engineering design synthesis* (Antonsson, E. K. and Cagan, J., eds.), pp. 228–269, Cambridge University Press, 2001.

[140] Stahovich, T. F., Davis, R., and Strobe, H., "Generating multiple new designs from a sketch," *Artificial Intelligence*, vol. 104, pp. 211–264, Sept. 1998.

[141] Steinberg, L. I., "Design as refinement plus constraint propagation: the vexed experience," *In Proceedings of the sixth national conference on artificial intelligence*, vol. 2, pp. 830–835, July 1987.

[142] Sternberg, R. J., "Component processes in analogical reasoning," *Psychological Review*, vol. 84, no. 4, pp. 353–378, 1977. Holyoak K. J. (ed.).

[143] Sussman, G. J., "Electrical design: a problem for artificial intelligence research," *In Proceedings of the 5th international joint conference on artificial intelligence*, vol. 2, pp. 894–899, Aug. 1977.

[144] Suwa, M., Purcell, T., and Gero, J., "Macroscopic analysis of design processes based on a scheme for coding designers' cognitive actions," *Design Studies*, vol. 19, pp. 455–483, Oct. 1998.

[145] Suwa, M. and Tversky, B., "What do architects and students perceive in their design sketches? a protocol analysis," *Design Studies*, vol. 18, pp. 385–403, Oct. 1997.

[146] Szykman, S., Sriram, R. D., Bochenek, C., Racz, J. W., and Senfaute, J., "Design repositories: engineering design's new knowledge base," *IEEE Intelligent Systems and their Applications*, vol. 15, no. 3, pp. 48–55, 2000.

[147] Tero, A., Takagi, S., Saigusa, T., Ito, K., Bebber, D. P., Fricker, M. D., Yumiki, K., Kobayashi, R., and Nakagaki, T., "Rules for biologically inspired adaptive network design," *Science*, vol. 327, pp. 439–442, Jan. 2010.

[148] Thagard, P., Holyoak, K. J., Nelson, G., and Gochfeld, D., "Analog retrieval by constraint satisfaction," *Artificial Intelligence*, vol. 46, pp. 259–310, Dec. 1990.

[149] The Owl Pages, "Leading edge of barn owl feather," 2018. Last retrieved from https://www.owlpages.com/owls/articles.php?i=18 on August 19, 2018.

[150] Tillmann, N. and Schulte, W., "Unit tests reloaded: parameterized unit testing with symbolic execution," *IEEE Software*, vol. July/August, pp. 38–47, 2006.

[151] Tomiyama, T., "Intelligent computer-aided design systems: Past 20 years and future 20 years," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, vol. 21, pp. 27–29, Jan. 2007. 20th Anniversary Issue.

[152] Tseng, I., Moss, J., Cagan, J., and Kotovsky, K., "The role of timing and analogical similarity in the stimulation of idea generation in design," *Design Studies*, vol. 29, pp. 203–221, May 2008.

[153] Umeda, Y., Takeda, H., Tomiyama, T., and Yoshikawa, H., "Function, behaviour, and structure," in *Applications of Artificial Intelligence in Engineering V* (Gero, J., ed.), vol. 1, pp. 177–193, Springer-Verlag, 1990. Berlin.

[154] Umeda, Y., Ishii, M., Yoshioka, M., Shimomura, Y., and Tomiyama, T., "Supporting conceptual design based on the function-behavior-state modeler," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, vol. 10, no. 44, pp. 275–288, 1996.

[155] Umeda, Y. and Tomiyama, T., "Functional reasoning in design," *IEEE Expert*, vol. 12, no. 2, pp. 42–48, 1997.

[156] Vattam, S., Helms, M., and Goel, A., "Nature of creative analogies in biologically inspired innovative design," *Proceedings of the Seventh ACM Conference on Creativity & Cognition*, Oct. 2009. Berkeley, California.

[157] Vattam, S. S., Goel, A. K., Rugaber, S., Hmelo-Silver, C. E., Jordan, R., Gray, S., and Sinha, S., "Understanding complex natural systems by articulating structure-behavior-function models," *Educational Technology & Society*, vol. 14, no. 1, pp. 66–81, 2011.

[158] Vattam, S. S., Helms, M., and Goel, A. K., "Biologically inspired design: A macrocognitive account," *ASME 2010 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference. 22nd International Conference on Design Theory and Methodology; Special Conference on Mechanical Vibration and Noise*, vol. 5, pp. 129–138, Aug. 2010. Montreal, Quebec, Canada.

[159] Vattam, S. S., Helms, M. E., and Goel, A. K., "Compound analogical design: Interaction between problem decomposition and analogical transfer in biologically inspired design," in *Design Computing and Cognition '08* (Gero, J. S. and Goel, A. K., eds.), pp. 377–396, Dordrecht: Springer, 2008.

[160] Vattam, S., *Interactive Analogical Retrieval: Practice, Theory and Technology*. PhD thesis, Georgia Institute of Technology, 2012.

[161] Vattam, S., Helms, M. E., and Goel, A. K., "A content account of creative analogies in biologically inspired design," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, vol. 24, pp. 467–481, 2010.

[162] Vattam, S., Wiltgen, B., Helms, M., Goel, A. K., and Yen, J., "DANE: Fostering creativity in and through biologically inspired design," *First International Conference on Design Creativity*, pp. 115–122, Nov. 2010. Kobe, Japan.

[163] Vincent, J. and Mann, D., "Systematic technology transfer from biology to engineering," *Philosophical Transactions of the Royal Society of London*, vol. 360, no. 1791, pp. 159–173, 2002.

[164] Vincent, J. F. V., Bogatyreva, O. A., Bogatyrev, N. R., Bowyer, A., and Pahl, A.-K., "Biomimetics: its practice and theory," *Journal of the Royal Society Interface*, vol. 3, pp. 471–482, Aug. 2006.

[165] Visser, W., "More or less following a plan during design: opportunistic deviations in specification," *International journal of man-machine studies*, vol. 33, pp. 247–278, Sept. 1990. Special issue: What programmers know.

[166] Visser, W., "Two functions of analogical reasoning in design: a cognitive-pscyhology approach," *Design Studies*, vol. 17, pp. 417–434, Oct. 1996.

[167] von Gleich, A., Pade, C., Petschow, U., and Pissarskoi, E., *Potentials and Trends in Biomimetics*. Berlin Heidelberg: Springer-Verlag, 2010.

[168] Wanieck, K., Fayemi, P.-E., Maranzana, N., Zollfrank, C., and Jacobs, S., "Biomimetics and its tools," *Bioinspired, Biomimetic and Nanobiomaterials*, vol. 6, no. 2, pp. 53–66, 2017.

[169] Welch, R. V. and Dixon, J. R., "Guiding conceptual design through behavioral reasoning," *Research in Engineering Design*, vol. 6, pp. 169–188, Sept. 1994.

[170] Williams, M. L., Ertas, A., and Tate, D., "Using stochastic multicriteria acceptability analysis in biologically inspired design as a multidisciplinary tool to assess biology-to-engineering transfer risk for candidate analogs," *Journal of Mechanical Design*, vol. 136, p. 111107, Oct. 2014.

[171] Wills, L. M. and Kolodner, J. L., "Towards more creative case-based design systems," *In proceedings of the twelfth AAAI national conference on artificial intelligence (AAAI'94)*, pp. 50–55, Aug. 1994.

[172] Wiltgen, B., Goel, A. K., and Vattam, S., "Representation, indexing, and retrieval of biological cases for biologically inspired design," *ICCBR 2011: Case-Based Reasoning Research and Development*, pp. 334–347, 2011.

[173] Woodford, C., "Electric toothbrushes," 2007/2014. Last retrieved from http://www.explainthatstuff.com/electrictoothbrush.html on May 12, 2018.

[174] Xtext, "XText - language engineering made easy!," 2018. Last retrieved from https://www.eclipse.org/Xtext/ on May 13, 2018.

[175] Yaner, P. W. and Goel, A. K., "Visual analogy: Viewing analogical retrieval and mapping as constraint satisfaction problems," *Applied Intelligence*, vol. 25, no. 1, pp. 91–105, 2006.

[176] Yaner, P. W. and Goel, A. K., "Analogical recognition of shape and structure in design drawings," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, vol. 22, no. 2, pp. 117–128, 2008. Spring.

[177] Yen, J., Helms, M., Goel, A., Tovey, C., and Weissburg, M., "Adaptive evolution of teaching practices in biologically inspired design," in *Biologically Inspired Design* (Goel, A., McAdams, D., and Stone, R., eds.), pp. 153–199, London: Springer, 2014.

[178] Yen, J., Helms, M., Vattam, S., and Goel, A., "Evaluating biological systems for their potential in engineering design," *Advances in Natural Science*, vol. 3, pp. 1–14, Sept. 2010. In Proceedings of the 3rd International Conference on Bionics Engineering. Zhuhai, China.

[179] Zhang, W. Y., Tor, S. Y., and Britton, G. A., "Managing modularity in product family design with functional modeling," *The International Journal of Advanced Manufacturing Technology*, vol. 30, pp. 579–588, Oct. 2006.

[180] Zhao, F. and Maher, M. L., "Using analogical reasoning to design buildings," *Engineering with Computers*, vol. 4, no. 3, pp. 107–119, 1988.