Illustration, Explanation and Navigation of Physical Devices and Design Processes

Nathalie Grué
nath@cc.gatech.edu

GIT-CC-94/26

A Masters Thesis
Presented to
The Academic Faculty

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in Computer Science

Georgia Institute of Technology
May 1994

This work has been supported by grants from Georgia Tech's Educational Technology Institute, College of Computing, Graphics, Visualization and Usability Center, and Cognitive Science program. In addition, it has been indirectly supported by the National Science Foundation (grant IRI-92-10925), the Office of Naval Research (contract N000/4-92-J-1234), and Northern Telecom.

ILLUSTRATION, EXPLANATION AND NAVIGATION OF PHYSICAL DEVICES AND DESIGN PROCESSES

ACKNOWLEDGMENTS

This work and my master's studies in general would not have been possible without the efforts and assistance of many people to whom I will always be in debt.

First, I thank Dr. Ashok Goel. Dr. Goel has been an outstanding advisor. He provided me with the opportunity to conduct this research, the funding for my master's studies and an excellent group of researchers with whom to collaborate. Dr. Goel's advice and constructive criticisms have been very significant to the progress of this work.

Dr. T. Govindaraj and Dr. Margaret Recker contributed in setting the goals of this research and advising on its progress. Their course on educational technology stimulated my interest in intelligent tutoring systems and provided many ideas for the development of CANAH-CHAB.

Dr. Mark Guzdial provided interesting advices on the use of visualization tools to improve the interface of CANAH-CHAB.

Drs. Goel, Govindaraj, Recker and Guzdial provided challenging critiques on earlier drafts of this thesis. Their constructive criticisms helped to make explicit the contributions and limitations of this research. They also provided detailed comments that helped to significantly improve the presentation of this thesis. Drs James Foley and John Stasko provided feedback on CANAH-CHAB's interface.

The research reported in this thesis has immensely benefited from input by several people as well. Mr. Andrés Gómez de Silva Garza played an important role in the implementation of CANAH-CHAB. His contributions have been especially significant in the development of the illustrations of the design processes. Mr. Michael Donahoo also helped with some aspects of the implementation of CANAH-CHAB's explanatory capabilities.

Ms. Eleni Stroulia provided help in expressing the design processes of KRITIK2 in the language of Structure-Behavior-Function models. She and Mr. Sambasiva Bhatta provided very useful help and advice during my two years of studies and the conduct of this research.

This work has been supported by grants from Georgia Tech's Educational Technology Institute, College of Computing, Graphics, Visualization and Usability Center, and Cognitive Science program. In addition, it has been indirectly supported by the National Science Foundation (grant IRI-92-10925), the Office of Naval Research (contract N000/4-92-J-1234), and Northern Telecom.

The friends I have come to know and work with during my two years at the College of Computing are too numerous to mention here. Nonetheless, they are indirectly responsible for much of my work and education while at Georgia Tech. Ms. Fabienne Derain provided practical advice and support during my studies at Georgia Tech. Mr. Vincent Garnier and Mr. Frédéric Vindreau have helped me time and time again to unwind and relax after a long day.

Finally, I am very grateful for the patience and support of my family. I am thankful to my brother, Bruno Grué, who initiated my interest in Computer Science, and to my sisters, Line Grué for sharing her interest in education and Maryse Piquiaud, for pushing me to go on with my studies. I am especially thankful to my parents, Maurice and Odette Grué, for their unlimited faith in me even though they were never really sure of what I was studying during the past two years. My parents are probably not aware that they supported me way beyond the call and duty of a parent, and that many of my accomplishments are attributed to their influence, role model, and never ending love and patience.

Contents

LIST OF FIGURES

SUMMARY

This work addresses three issues in intelligent tutoring systems. First, how to explain and illustrate the functioning of physical devices. This capability is useful for understanding and using past designs. The second issue is how to illustrate and explain the reasoning of an expert designer. This capability is often required for supporting learning by observation. Finally, might the same kinds of representations and processes support these two capabilities? This thesis explores how Structure-Behavior-Function models of physical devices and Structure-Behavior-Function meta models of an autonomous design system help to address these issues. The Structure-Behavior-Function model of a device (or a system) explicitly specifies its structure, its function and its internal behaviors that map its structure into its function. This research has resulted in a system called CANAH-CHAB that provides an interactive graphical interface to an autonomous design system called KRITIK2. It illustrates and explains both the physical devices in KRITIK2's memory and the processes by which it generates new design. In addition, CANAH-CHAB enables a user to navigate KRITIK2's memory and access the stored designs.

CHAPTER I

INTRODUCTION

1.1 Overview

This work explores three issues in intelligent tutoring systems. First, how to explain and illustrate the functioning of physical devices. This capability is useful for understanding and using past designs. The second issue is how to illustrate and explain the reasoning of an expert designer. This capability is often required for supporting learning by observation. Finally, might the same kinds of representations and processes support these two capabilities? The goal of this research is to develop an enabling technology for addressing these issues. In particular, this thesis investigates how Structure-Behavior-Function models of physical devices and Structure-Behavior-Function meta models of an autonomous design system can help to address these issues. The Structure-Behavior-Function (SBF) model of a device (or a system) explicitly specifies its structure, its function and its internal behaviors that map its structure into its function.

This research has resulted in a system called CANAH-CHAB[1] that provides an interactive graphical interface to an existing autonomous design system called KRITIK2. CANAH-CHAB illustrates the conceptual design of engineering devices such as heat-exchangers and electrical circuits. It illustrates and explains both the physical devices in KRITIK2's memory and the processes by which it generates new designs.

[1] "Canah" and "chab" are Mayan words that mean "learning" and "designing," respectively. The system was developed jointly with Andrés Gómez de Silva Garza.

In addition, Canah-chab enables a user to navigate through Kritik2's memory and access the stored designs.

1.1.1 Background

Computer Aided Design (CAD) tools and systems enable the representation and visualization of the structure and geometry of complex design objects such as car engines and airplanes ([Majumder, Fulton and Shilling 1990]). They can provide different views of the shape and the geometry of these devices, and the parts and components they are made of. For example, they offer two dimensional and three dimensional views of these objects, enabling a good visualization of the objects as they are, or would be, in the real world.

However, CAD tools support only some "low-level" tasks such as drafting and drawing, geometric and solid modeling, numerical computing and constraint propagation. While they provide a limited degree of flexibility in sketching, analyzing and refining design solutions, they are useful mainly for configuration design and design layout but not for *conceptual* design.

These observations have led to several proposals for the development of computer-based tools and environments for interactively supporting conceptual design. For example, Fischer *et al.* have developed the Janus system for supporting indirect collaboration among members of design teams engaged in conceptual design [Fischer *et al.* 1992]. Fischer shows that design *construction kits* are necessary for good design but that they are not sufficient. In order to assist the user in constructing interesting and useful artifacts, construction kits also need to provide knowledge to distinguish "good" from "bad" designs. To help improve design problem solving, Janus provides access to the reasoning process that led to a past design. It uses the *issue-based method* [Rittel 1972] to describe the design rationale, and the knowledge base includes a catalog of designs.

In an experimental system for supporting conceptual design called BOGART, Mostow [1989] explored the issue of the reuse of the process of design instead of merely the product of design. BOGART uses *derivational analogy* [Carbonell *et al.* 1989] for supporting circuit (re-)design by mechanically *replaying* the recorded history of design decisions made in a previous design. It has been used to help design simple NMOS digital circuits, and its techniques have also been applied to mechanical design and algorithm design.

1.1.2 Motivation

The long term goal of this research is to build an interactive design and learning environment that can support "high-level" tasks of conceptual design, such as the generation and analysis of designs in engineering domains, and also support the learning of these tasks. A general issue in the development of such a system is to determine the kinds of knowledge this environment should provide to the users in order to support conceptual design (and learning). Kolodner [1991, 1993] has suggested providing access to libraries of past design cases. The ARCHIE system [Pearce *et al.* 1992] was an early attempt in developing computer-based case libraries to support conceptual design of new office buildings. The system provides architects with a library of past design cases to help them in two subtasks of conceptual design: design generation and design critiquing. A design case in ARCHIE contains specifications of the goals, plans and outcomes of a specific design. One of the lessons learned from ARCHIE was that design problem solving seems to involve several types of knowledge in addition to that of design cases, e.g., design principles and guidelines are needed to generate new designs, and domain models are useful in critiquing proposed designs. The lessons learned from ARCHIE were taken into account in the ASKJEF project [Barber *et al.* 1992]. ASKJEF supports software engineers in designing human-machine interfaces. It provides software engineers access to a computer-based library that contains several types of knowledge such as design cases from the domain of human-machine interface design, design stories from other domains, design principles and guidelines, design objects, and prototypical designer errors. The knowledge and interface organization ideas of AskJef led to a new version of Archie called ArchieTutor [Goel *et al.* 1993]. The goal of ArchieTutor is to support design teaching in beginning architectural classes. The system provides access to knowledge of design cases, domain models and design principles and guidelines.

One of the key lessons from the Archie, AskJef and ArchieTutor projects is that it is not sufficient to provide access to a library of past design cases. In addition to a library of static cases, the environment needs to provide access to *dynamic knowledge* of (i) how the design works, i.e., how the structure of the design delivers its behavior, and (ii) how the design process works, i.e., how an expert designer creates and analyzes new designs. These lessons have led to the work described here.

1.1.3 Goals

This research describes three capabilities that an intelligent tutoring system should have in order to give user access to knowledge of physical devices and to make the knowledge and reasoning of an autonomous design system transparent and explicit to the users.

Illustration: This thesis describes the content of graphical illustrations useful for presenting dynamic knowledge about design objects and processes to the user. It shows how to illustrate devices so that their functioning is made explicit. It also shows how to illustrate the tasks and methods involved in the design activity and the knowledge used by an autonomous system.

Explanation: To improve the user's understanding of the design process, the intelligent tutoring system needs to generate explanations about the design system's reasoning and justifications of the methods and strategies it uses.

Navigation: In order for a user to acquire a good understanding of the functioning of a device, she should be able to browse through the different conceptual views of the device. The system also needs to support navigation through the system knowledge so that the user can explore the content and the organization of the system's memory.

The goal of this research is to develop an enabling technology for (i) exposing the user to dynamic knowledge of physical devices and design processes, (ii) generating explanations about the functioning of devices and about the design processes, and (iii) enabling the user to navigate through external representations of devices and design processes. Instead of providing the user with access to the reasoning process of a human expert designer, this work illustrates an autonomous design system.

One can adopt two views to the development of the above capabilities in intelligent tutoring systems: a cognitive view and a technological view. The former leads to explorations of the different kinds of illustrations and explanations that may help real users in their design and learning tasks. On the technological side, one can explore the representations and processing needed for illustrative and explanatory purposes. This work has focused entirely on the technological side, and addresses the following questions: (i) How to explain and illustrate the functioning of physical devices? (ii) How to illustrate and explain the design processes involved in an expert designer's reasoning? (iii) Do these abilities require fundamentally different representations and processes?

1.1.4 Hypotheses

This thesis explores the following hypotheses: (i) Structure-Behavior-Function models of physical devices [Goel 1989, 1991a, 1991b] enable illustration and explanation of how physical devices work; (ii) Structure-Behavior-Function meta models of reasoning processes [Stroulia and Goel 1993, 1994] enable illustration and explanation of how the design process (as instantiated in an autonomous design system) works; (iii) The successful use of SBF models and meta models for the two different abilities would indicate that the same kind of content theory, representation scheme, and presentation mechanism can be used to express knowledge about physical devices and design processes.

1.2 CANAH-CHAB

1.2.1 Product

CANAH-CHAB, the product of this research, is an interactive computing environment that uses an autonomous design system, KRITIK2 ([Goel 1991a, 1991b, 1992], [Goel and Chandrasekaran 1989, 1992], [Bhatta and Goel 1992, 1993]), as the design expert. It supports illustration, navigation and explanation of physical devices and of KRITIK2' s reasoning processes. It has been developed on a Sun Sparc Station using the Common Lisp Object System and the Garnet interface development tool ([Myers and Zanden 1992, Myers 1990]).

1.2.2 KRITIK2

Instead of developing a complete new autonomous system that would perform design, our work directly builds on KRITIK2. The design process performed by KRITIK2 and the physical devices it knows about and designs are illustrated and explained by CANAH-CHAB.

KRITIK2 is a cognitively-inspired autonomous design problem-solving and learning system that operates in the domain of engineering devices such as electrical circuits and heat exchangers. It takes as input the user's specification of the function that is desired, and provides as output a description of the structure of a device that produces the function. It uses multiple types of knowledge in solving design problems, for example, past design cases and Structure-Behavior-Function (SBF) models of how the specific devices stored in the design cases actually work. A design case is indexed by the functions delivered by the stored design and acts as an index into the SBF model of the stored design. Given a design problem, KRITIK2 solves it by first elaborating on the functional specification of the problem, and retrieving from its case memory a design that delivers a function similar to the desired one. It then identifies candidate modifications to the structure of the retrieved design by analyzing the SBF model of the design, executes the required modifications on the design structure to produce a candidate design, and revises the SBF model of the retrieved design to produce an SBF model for the candidate design. The system then evaluates the candidate design by qualitatively simulating the SBF model. If the design fails then KRITIK2 attempts to redesign it. If it succeeds, then KRITIK2 uses the SBF model of the new design as a causal explanation to learn the indices for storing the new design case and associated SBF model in memory, and stores them for potential reuse in the future.

A problem-solving task in this framework is specified by the information it takes as input and the information it produces as output. A task can be accomplished by one or more methods, each of which decomposes it into a set of simpler subtasks. A method is specified by the subtasks it sets up, the control it exercises over their processing, and the knowledge it uses. The subtasks into which a method decomposes a task can, in turn, be accomplished by other methods, or, if the appropriate knowledge and procedures are available, they can be solved directly. This enables KRITIK2 to opportunistically select a problem-solving method depending on the current subtask it is addressing. Since the system may select different methods for different subtasks, this enables KRITIK2 to integrate several reasoning strategies such as case-based and model-based reasoning and to shift from one strategy to another depending on the needs of the current subtask. KRITIK2's task structure for design is illustrated in Figure 1.1.

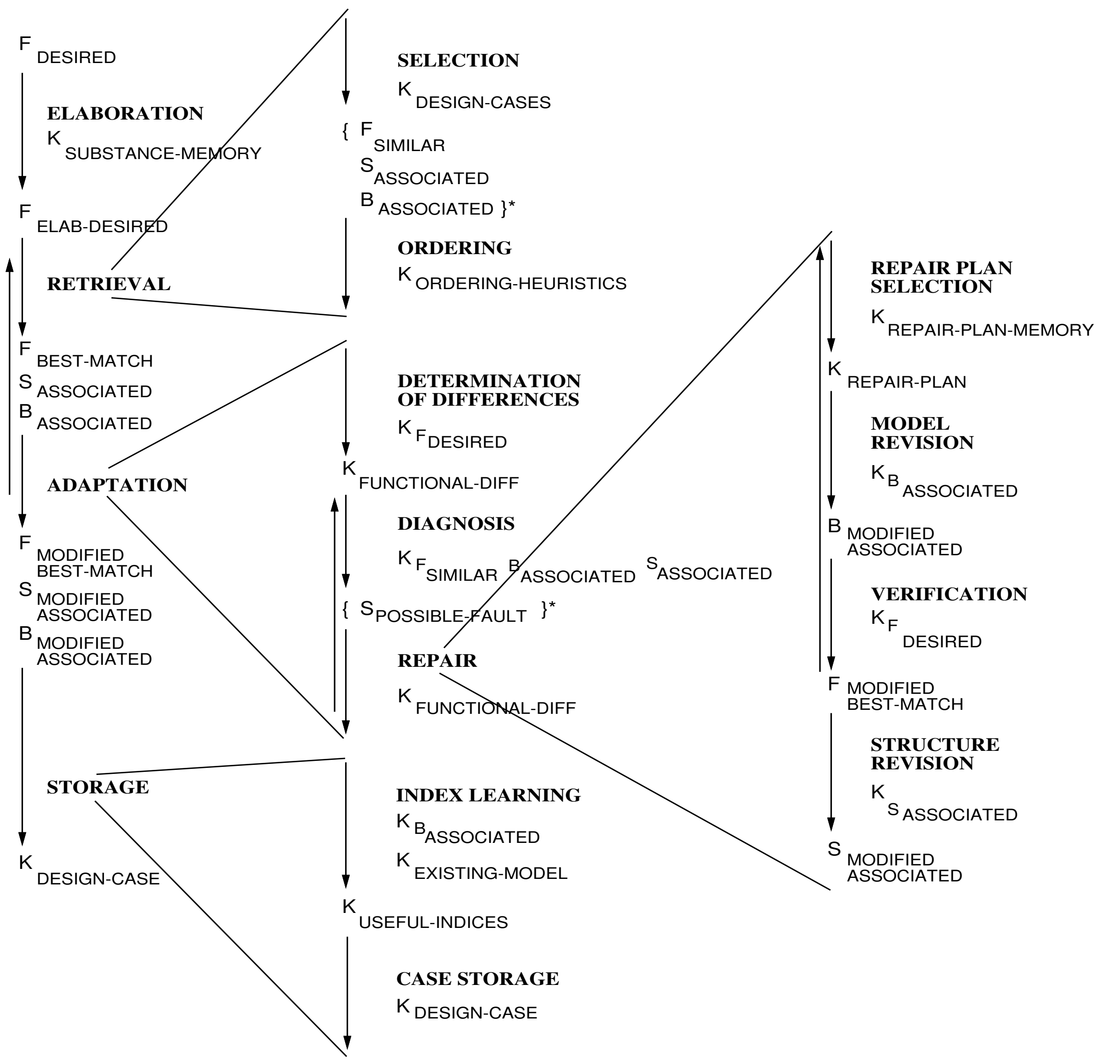


Figure 1.1: KRITIK2's Task Structure.

1.2.3 Architecture of Canah-chab

Canah-chab provides an interactive graphical interface to Kritik2. It has access to all the knowledge of Kritik2. In particular, it uses the SBF models of physical devices defined in Kritik2 to graphically illustrate and explain the functioning of such design artifacts to the users. Further, it views Kritik2 as an abstract device and explicitly models its reasoning process. It uses these SBF meta models of Kritik2 to illustrate and explain how the system generates new designs. The architecture of Canah-chab and its relation to Kritik2 is illustrated in Figure 1.2.

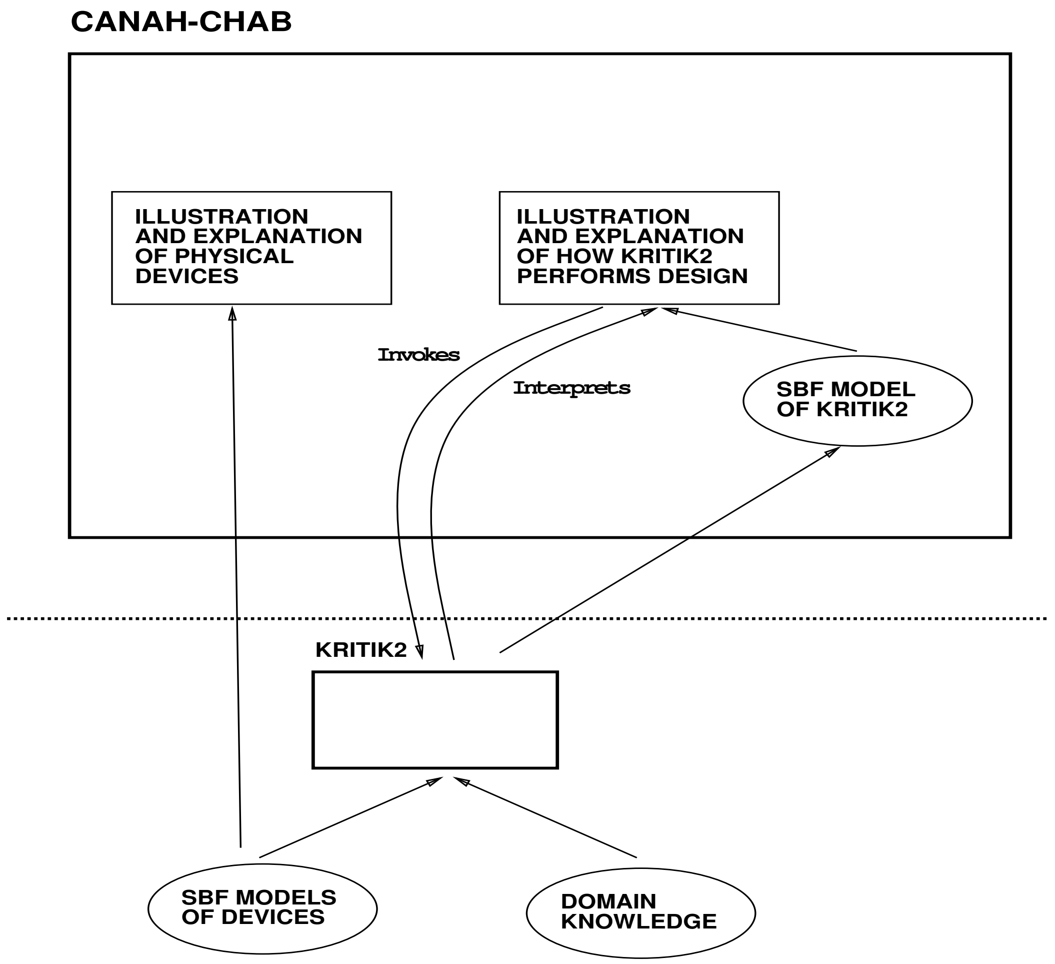

CANAH-CHAB
ILLUSTRATION AND EXPLANATION OF PHYSICAL DEVICES
ILLUSTRATION AND EXPLANATION OF HOW KRITIK2 PERFORMS DESIGN
Invokes
Interprets
SBF MODEL OF KRITIK2
KRITIK2
SBF MODELS OF DEVICES
DOMAIN KNOWLEDGE

Figure 1.2: Canah-chab's Architecture

Canah-chab invokes Kritik2 to provide the user with an illustration of how the latter performs a design task. It graphically illustrates a set of problem-solving examples. The user may observe the design process and the knowledge used by Kritik2 in addition to accessing the illustrations and explanations of past devices in the context of these illustrative examples. Canah-chab also supports navigation through Kritik2's memory of physical devices.

1.2.4 Examples

1.2.4.1 Design Objects

Canah-chab provides graphical representations of both the devices retrieved from Kritik2's memory, and the new devices created by Kritik2. The design objects are physical devices such as electrical circuits, heat exchangers, etc. To illustrate the functioning of such devices, Canah-chab provides the user with a screen representing the function it achieves. Figure 1.3 is an example of Canah-chab's screen illustrating the function of an electrical circuit. The function of this device is to produce a certain amount of light.

The means by which the function of a device is achieved is explained in the SBF model of the device by its internal causal behaviors. Figure 1.4 shows an illustration by Canah-chab of the main behavior of the electrical circuit that produces light. Figure 1.5 shows the secondary behavior of this device. It represents the behavior of the electricity in this circuit.

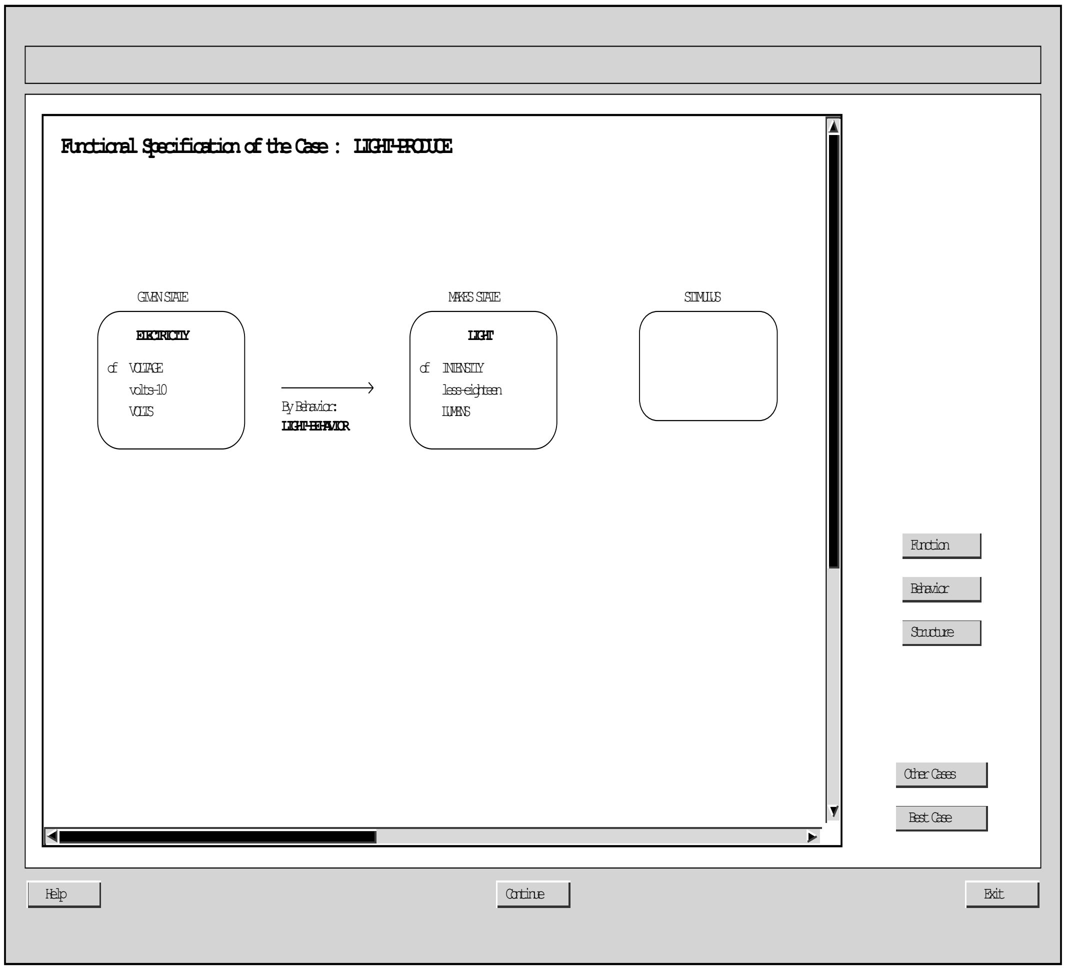
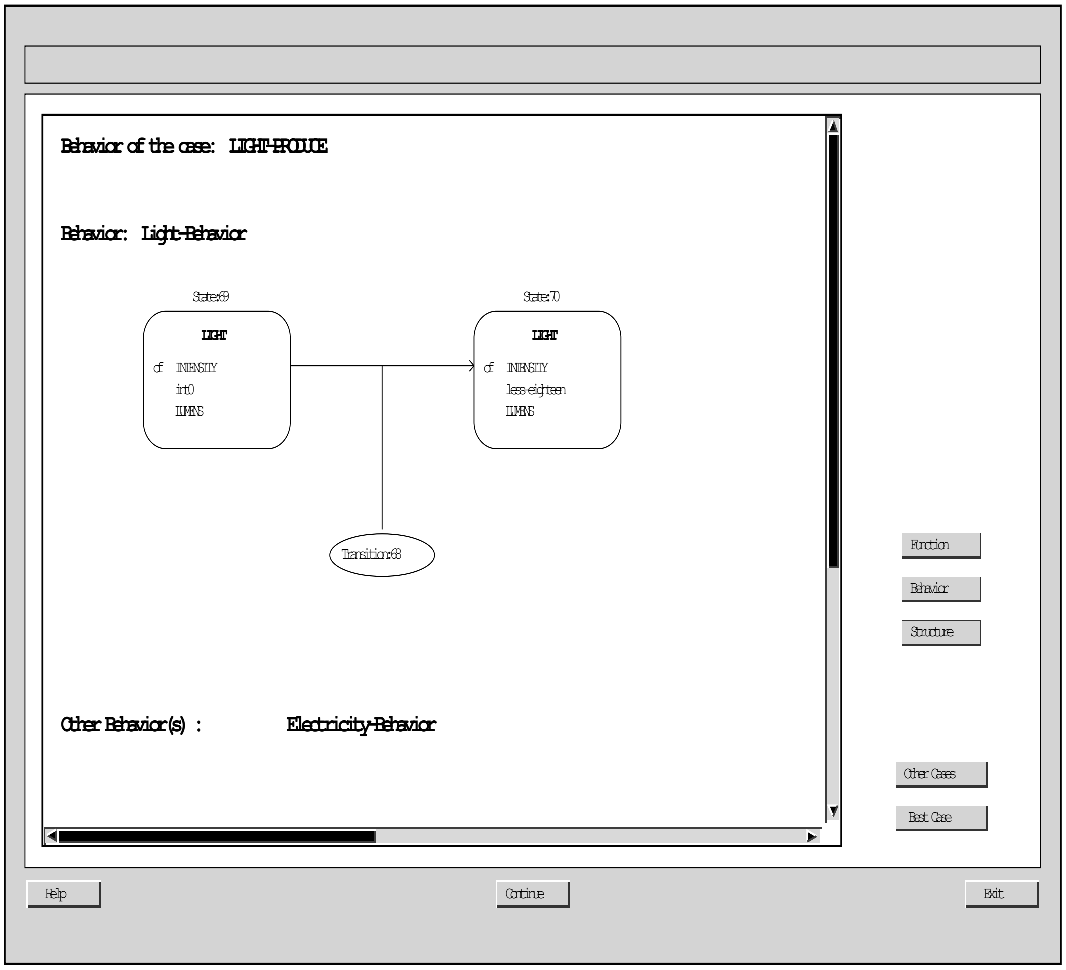


Figure 1.3: The Function of an Electrical Circuit

1.2.4.2 Design Processes

CANAH-CHAB currently illustrates graphically the domain and process knowledge of KRITIK2. The reasoning of KRITIK2, while it is solving a problem, is captured in its SBF meta models in terms of tasks, methods, and domain concepts. These are illustrated by CANAH-CHAB in screens identifying the tasks that KRITIK2 is currently performing while it is solving a problem. Figure 1.6 shows the first task screen in CANAH-CHAB. It presents to the user the design task in terms of the Case-Based Reasoning method it is going to use and the subtasks that are set up by this method.


Behavior of the case: LIGHT-PRODUCE

Behavior: Light-Behavior

Other Behavior(s) : Electricity-Behavior

Figure 1.4: The Light Behavior of an Electrical Circuit

Figure 1.6 shows the four subtasks, Problem Elaboration, Case Retrieval, Design Adaptation, and Case Storage, that Kritik2 performs, along with their respective procedures or methods. It also shows the input, here a functional specification, to the first subtask.

Canah-chab provides a set of screens presenting the user with the input and output of the subtasks and uses highlighting features to notify to the user which tasks have already been performed, what is the current task and what subtasks are left. Figure 1.7 shows the task screen once Problem Elaboration and Case Retrieval

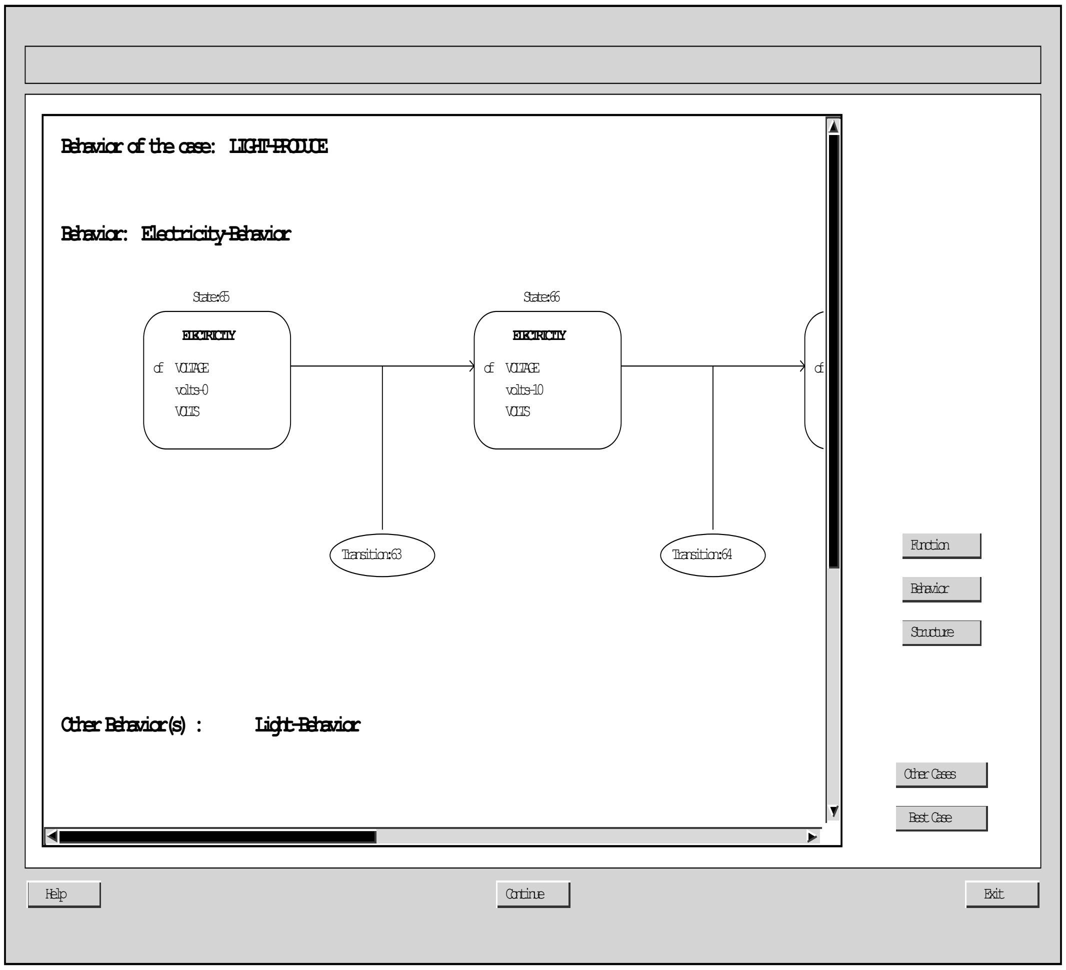


Figure 1.5: The Electricity Behavior of an Electrical Circuit

have been performed, with their respective inputs and outputs. Figure 1.7 shows that Design Adaptation is done by applying the Model-Based Adaptation method.

Figure 1.8 shows the representation of the subtasks set up by this method. It illustrates a deeper level of KRITIK2's task decomposition. In the same way, the Design Adaptation's subtask of Repair is done by applying the Model-Based Method shown in Figure 1.9. Once the low level task has been performed, CANAH-CHAB presents the result at a higher level of decomposition. Figure 1.10 shows the resulting knowledge and overall tasks performed by KRITIK2 to produce a new design.

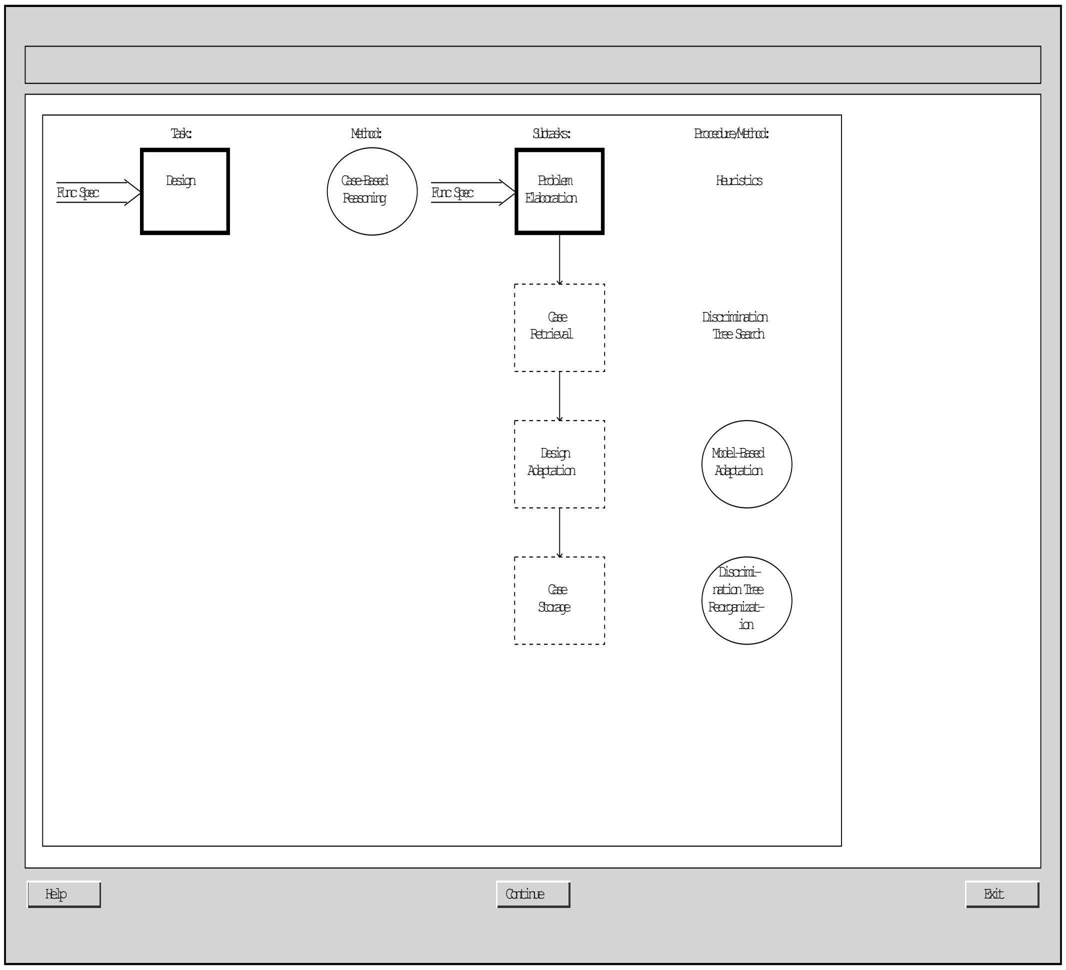


Figure 1.6: The Overall Design Task

Canah-chab not only illustrates the design objects and the design processes, it also illustrates the domain objects used by Kritik2 and defined in the SBF meta model of the system's reasoning as *domain concepts*. An example of this kind of object is the memory of past design cases used by Kritik2. Figure 1.11 shows Canah-chab's illustration of Kritik2's memory of design cases.

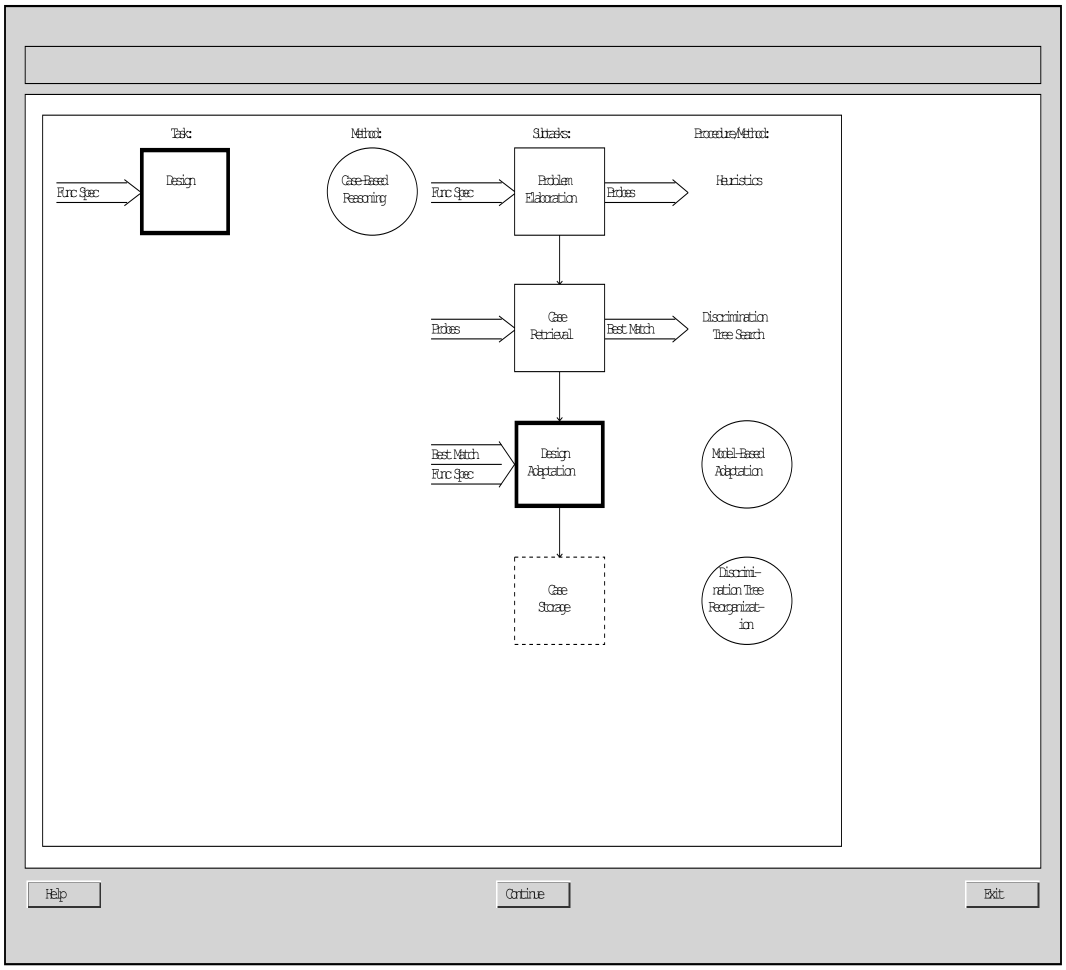


Figure 1.7: The Design Task before Design Adaptation

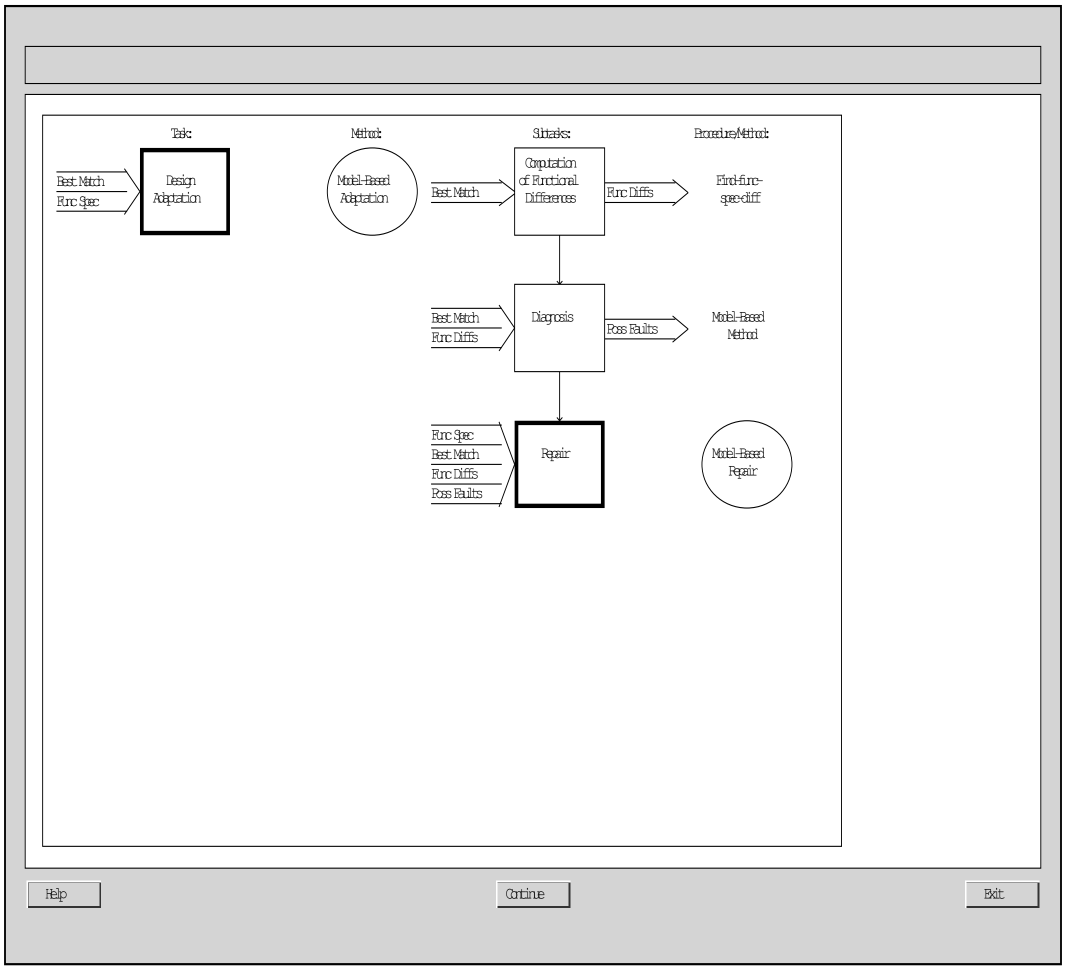


Figure 1.8: The Incomplete Design Adaptation Task

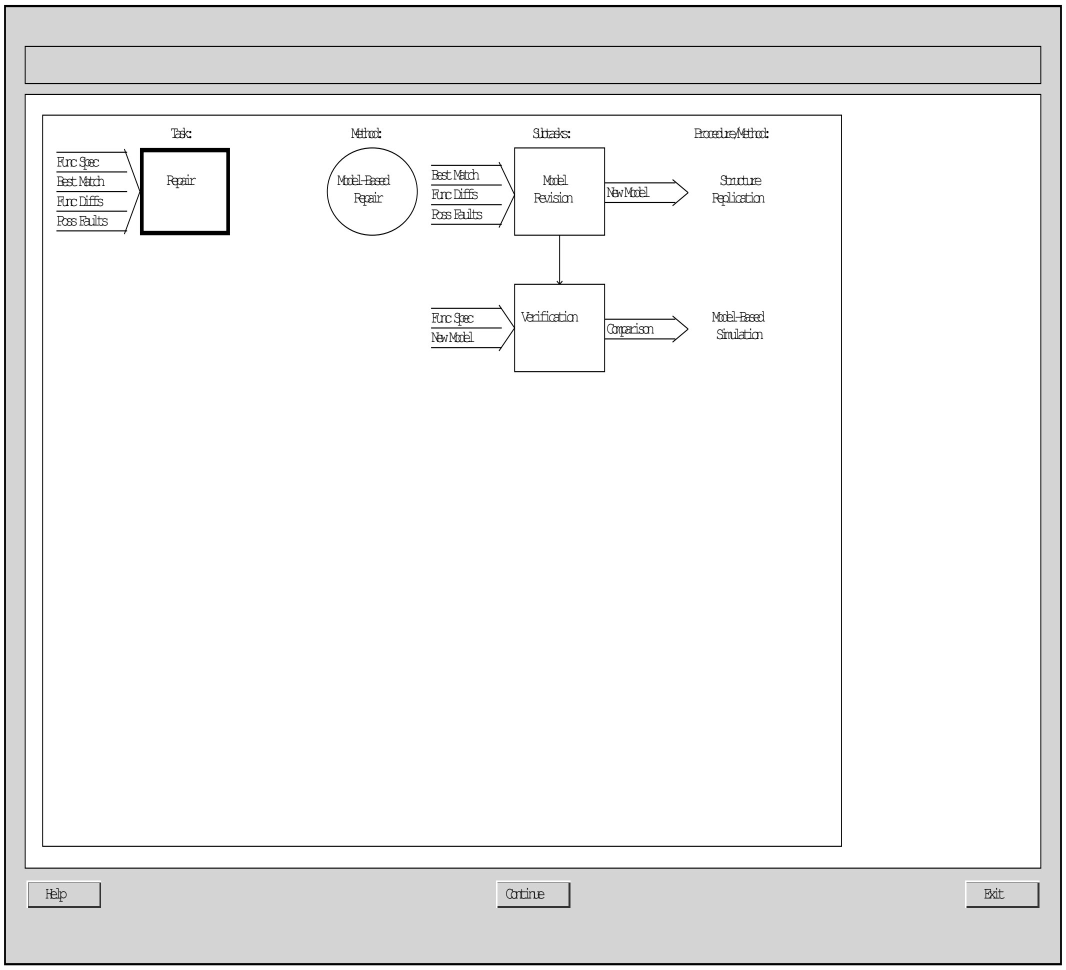


Figure 1.9: The Repair Task

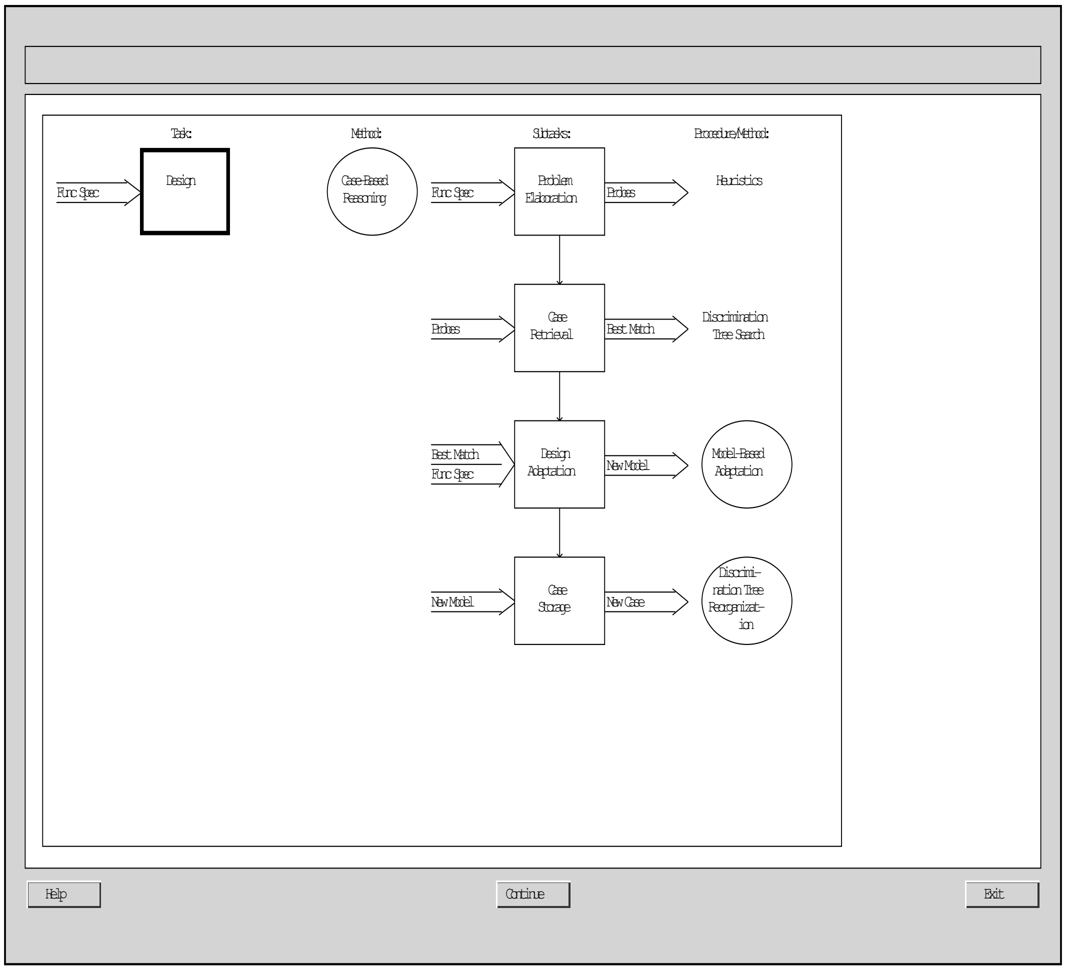


Figure 1.10: The Complete Design Task

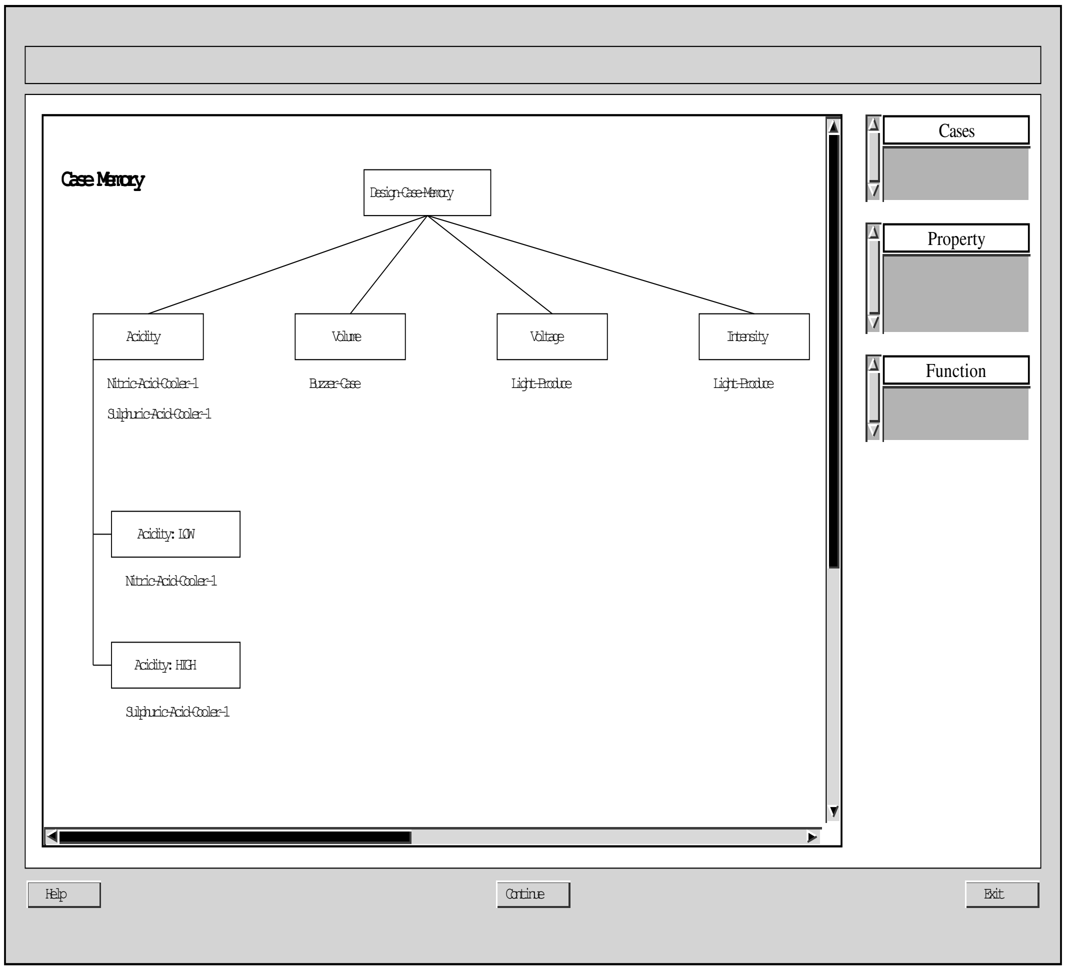


Figure 1.11: Kritik2's Memory of Devices

CHAPTER II

SBF MODELS AND META MODELS

This thesis describes the use of SBF models of physical devices to illustrate and explain the functioning of design artifacts and the use of SBF meta models of an autonomous design system to illustrate and explain the design processes to a user in an intelligent tutoring system. This chapter of the thesis discusses the knowledge contained in these models, and their representation and organization.

2.1 The SBF Models of Design Objects

Design Object: For each design in its memory, KRITIK2 comprehends how it works in terms of an SBF model. The SBF model of a design case specifies the functions of the device, its structure, and the internal causal processes that explain how its structure accomplishes its functions. The language of SBF models is one kind of *functional representation scheme* for representing knowledge about the functioning of a device [Sembugamoorthy and Chandrasekaran 1986]. Figure 2.1 shows the general Structure-Behavior-Function language for physical devices, as it is used in KRITIK2.

The function of the device is the observable behavior which was intended by the designers of the artifact. A device may exhibit many behaviors, some of which were not meant by its designers but are either necessary to support the functions of the device or are simply spurious. The causal model of the device consists of a set of internal causal processes that describe the device operation resulting in the

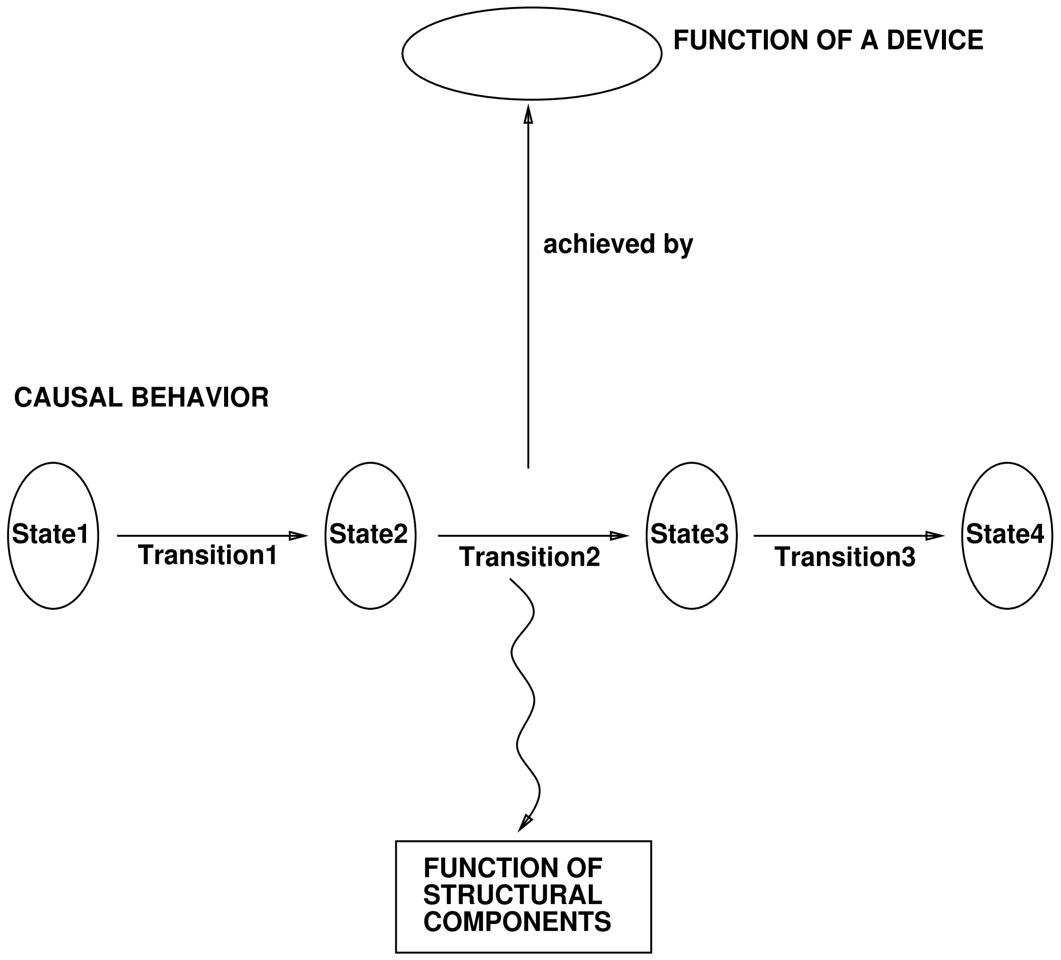


Figure 2.1: Structure-Behavior-Function Language for Physical Devices

accomplishment of its function. The causal model of a design case constitutes an explanation for the accomplishment of its function, in terms of the functioning of its structural elements and the laws of physics. Finally, the structure of the device consists of the structural elements of the device and their structural relations. Figure 2.2 shows the SBF model of an electric circuit in KRITIK2's memory[1].

[1]This graph, representing the SBF model of an electrical circuit in KRITIK2's memory, was taken from [Bhatta and Goel 1993].

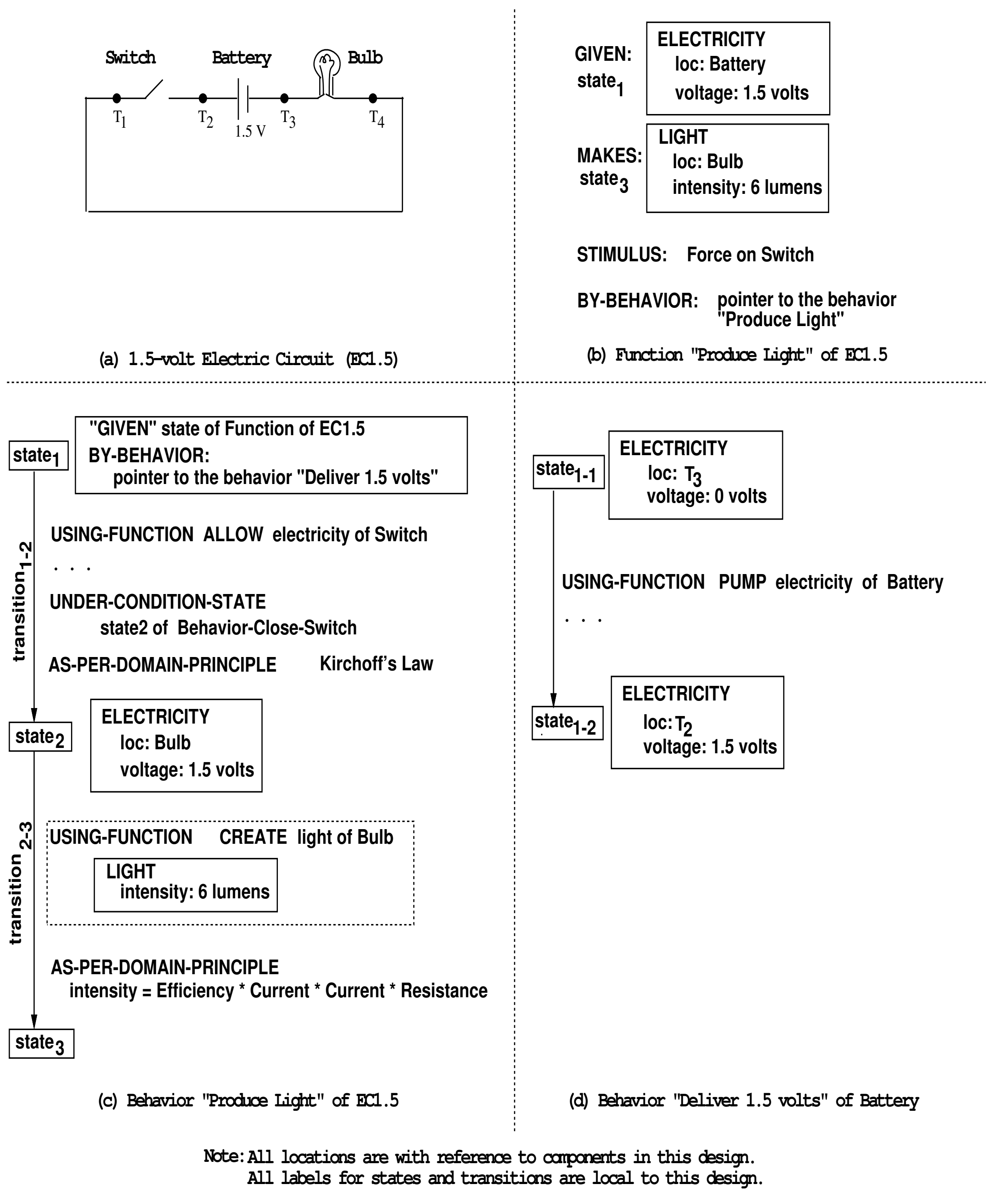

Switch
Battery
Bulb
T1
T2
1.5 V
T3
T4
(a) 1.5-volt Electric Circuit (EC1.5)
GIVEN: state1
ELECTRICITY
loc: Battery
voltage: 1.5 volts
MAKES: state3
LIGHT
loc: Bulb
intensity: 6 lumens
STIMULUS: Force on Switch
BY-BEHAVIOR: pointer to the behavior "Produce Light"
(b) Function "Produce Light" of EC1.5
state1
"GIVEN" state of Function of EC1.5
BY-BEHAVIOR:
pointer to the behavior "Deliver 1.5 volts"
transition1-2
USING-FUNCTION ALLOW electricity of Switch
. . .
UNDER-CONDITION-STATE
state2 of Behavior-Close-Switch
AS-PER-DOMAIN-PRINCIPLE Kirchoff's Law
state2
ELECTRICITY
loc: Bulb
voltage: 1.5 volts
transition2-3
USING-FUNCTION CREATE light of Bulb
LIGHT
intensity: 6 lumens
AS-PER-DOMAIN-PRINCIPLE
intensity = Efficiency * Current * Current * Resistance
state3
(c) Behavior "Produce Light" of EC1.5
state1-1
ELECTRICITY
loc: T3
voltage: 0 volts
USING-FUNCTION PUMP electricity of Battery
. . .
state1-2
ELECTRICITY
loc: T2
voltage: 1.5 volts
(d) Behavior "Deliver 1.5 volts" of Battery
Note: All locations are with reference to components in this design.
All labels for states and transitions are local to this design.

Figure 2.2: SBF Model of an Electrical Circuit in KRITIK2's Memory.

Structure: The SBF models are based on a *component-substance ontology* similar to the ontology used earlier in the *consolidation method* for deriving the behaviors of a device from the behavioral interactions between its structural components [Bylander and Chandrasekaran 1985, Bylander 1991]. Thus, physical devices are viewed as consisting of a set of connected components, through which substances flow. As the device substances flow, they are modified by the functioning of the components through which they flow. These substance transformations and their effects on the components functioning give rise to the device's intended functions.

The structure of the device is described hierarchically in terms of its constituent structural elements. The structural elements of a device may be primitive components, such as a battery, or they may be complex structures, such as an air-conditioning unit, which can themselves be further described in terms of their structural elements. Each structural element, except for the overall structure of the device, points to another structural element of which it is a part. In addition to the *part of* relation between structural elements, other structural relations, such as connectivity, containment, and spatial proximity are explicitly represented in the structure schema. Finally, each structural element points to the set of behavioral transitions in the device's causal model that affect the particular element or are affected by it.

The primitive structural elements of a device are its components and substances. The primitive components of a device are described in terms of their structural properties, other components of which they can be viewed as particular instantiations, and their modes of operation. For each different mode of operation, a primitive component exhibits a set of primitive functions.

The substances of a device are described in terms of properties and other substances of which they are instances, and they can be physical, such as water, or abstract, such as heat.

Functions: Physical devices can perform several different kinds of functions, such as control functions, prevention functions, maintenance functions, and transformation functions. Currently, KRITIK2 deals with transformation functions, which are described in terms of an initial and a final behavioral state. In addition to the two behavioral states, the description of the device function includes a pointer to the internal causal behavior responsible for the transformation. The function description also includes the stimulus from the external environment which triggers the transformation process, and sometimes, conditions, external to the device, which are necessary for the device functioning.

Internal Causal Behaviors: As substances flow through the device components, their properties may change and also the mode of operation of the components may change. The sequence of these states and state transitions constitute the *internal causal processes* or *internal causal behaviors* of the device. As it has been discussed in the previous paragraph, the intended function of the device acts as an index to the internal behavior which explains the transformation from the initial state, $given(F)$, of the function to its final state, $makes(F)$. This behavior is called the primary behavior of the device. For each device, however, there may be other, internal behaviors, supporting the primary one, which are equally important for the device functioning, and which may also result in observable phenomena which are not part of the device function. Each behavior explains the qualitative changes in one substance of the device.

In SBF models, knowledge of causal behaviors is represented as directed acyclic graphs (DAGS), where the behavioral states constitute the nodes of the graph and the state transitions its edges.

Behavioral State: A behavioral state is a partial description of the state of an element of the device, either a component or a substance, at a particular point in the device functioning.

A substance behavioral state specifies the previous and next states in the behavior sequence, and also the enabling transition, the transition which results in the state, and the transition which follows the state.

If it is a substance state, then the schema representing it specifies the type of the substance being described, the location of the substance at the time, and a set of property-value-unit tuples. The type of a substance is a pointer to a conceptual memory of substances which contains all the different kinds of substances KRITIK2 knows about.

If it is a component state, the state description specifies the type of the component, its mode of operation at the time, and a set of parameter-value-unit tuples. The type of a component is a pointer to a prototypical component in KRITIK2's conceptual memory of components.

Behavioral State Transition: A behavioral state transition is a partial description of a transformation of some device element during the functioning of the device. A behavioral transformation of a device element may be explained at several levels of abstraction and detail. Thus, the state-transition schema may include a pointer to another behavior, which explains in more detail the transformation described by that transition. For example, a *by-behavior* pointer results in the hierarchical organization of the device internal behaviors. In addition to pointing to a more detailed behavior, a state transition may explain a behavioral transformation in terms of the function of a structural element of the device, or in terms of a physics principle. Moreover, the transition schema may be annotated with qualitative equations describing the changes to the values of different substance properties and component parameters because of the transition.

A state transition may be conditioned upon the co-occurrence of specific behavioral states in the device, or the co-occurrence of other state transitions, or specific structural relations among the device elements. The explicit specification of such conditions is included in the state-transition schema. The language of SBF models provides a typology of conditions for describing the interactions of the different internal causal processes of a device: *under-condition-state, under-condition-state-transition, under-condition-structural-relation.*

The state-transition description also includes pointers to the behavioral states preceding and succeeding it, i.e., the behavioral state which enables the transition and the state into which the transition results. The state transitions, in addition to being causal transitions, capture an implicit temporal ordering of events in the device functioning. Since, in general, the cause temporally precedes the effect, the antecedent state temporally precedes the consequent state. Moreover, the conditions on the transition implicitly capture temporal co-occurrence, i.e., if a state transition is dependent upon another transition then they occur at the same time.

2.2 The SBF Meta Model of Domain and Process Knowledge

To help the users acquire some notions of design processes, the system has to communicate the design tasks and problem-solving methods. This is not possible without an explicit declarative representation of the domain and process knowledge used by the system. The previous section discussed the language of SBF models to represent the functioning of simple engineering devices in KRITIK2. Indeed, the same language can be used to capture the functioning of KRITIK2. In a different project called AUTOGNOSTIC, Stroulia and Goel [1993, 1994] have viewed problem solvers as abstract devices and how their functioning can be described in terms of the Structure-Behavior-Function (SBF) models that are used in KRITIK2 to model physical devices. Just like work on KRITIK2 led to the development of a SBF language for representing and organizing knowledge of the functioning of a physical device, similarly work on AUTOGNOSTIC has led to the development of a related SBF language for specifying knowledge of the reasoning tasks and methods employed by a problem solver in solving a problem. Figure 2.3 shows the general framework of Structure-Behavior-Function meta models. In CANAH-CHAB, this SBF language for problem solvers is

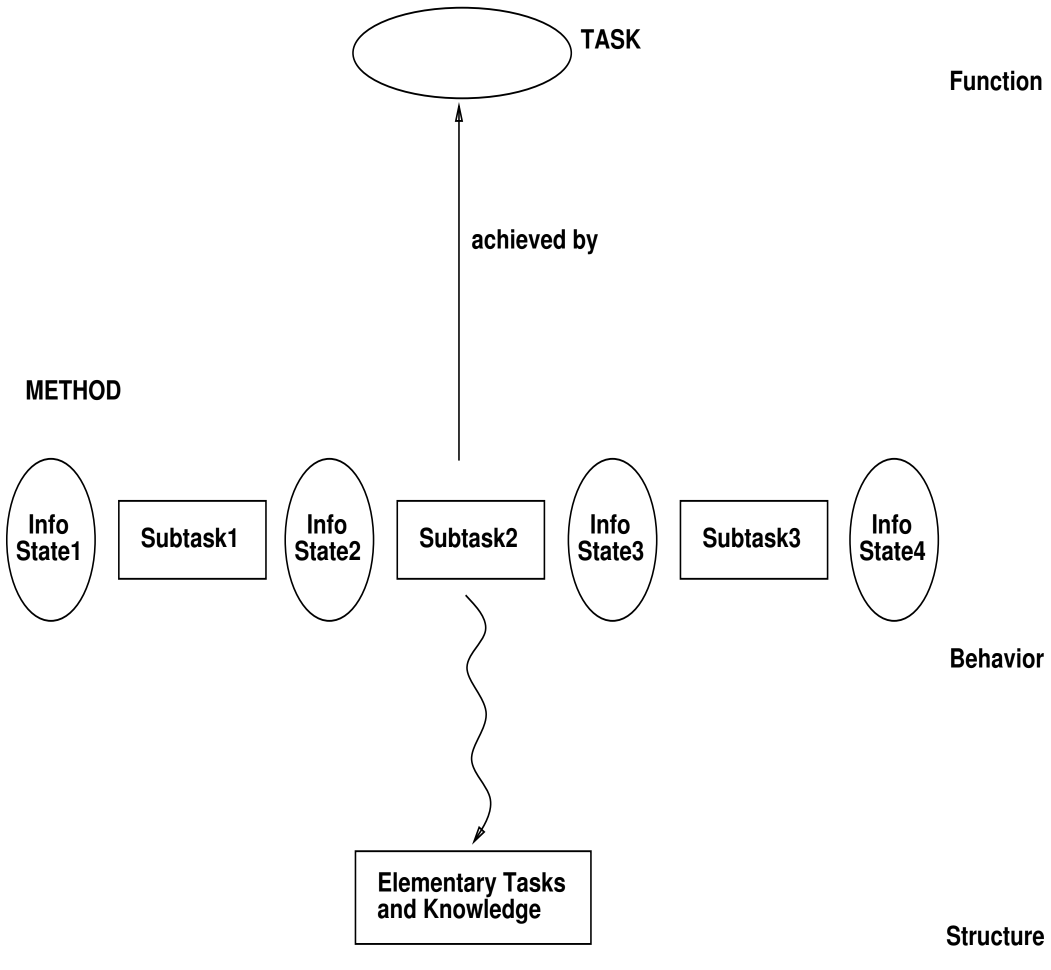


Figure 2.3: Structure-Behavior-Function Language for Problem Solvers

used to specify the design strategies that KRITIK2 uses. This way, the design of CANAH-CHAB takes advantage of the task-structure analysis of KRITIK2.

The vocabulary provided by the SBF semantics allows the representation of the domain knowledge and process knowledge of the system. The reasoning processes of KRITIK2 can be captured in the SBF language by defining the tasks and subtasks it sets up, the methods it uses and the objects from the domain that are involved.

Tasks: A task can be characterized in terms of its *input* and *output*, which are the substances that flow as arguments into the procedure that will carry out the task, the name of the *procedure* that will carry out the task, the list of methods *by* which the task can be solved, the *prototypical* (generic) task that this task is an instance of, and the conditions *under-condition* under which the task should be performed. Figure 2.4 shows the general semantic used to describe the tasks and subtasks performed by KRITIK2[2].

Methods: A method is characterized in terms of the task to which it is *applicable*, the *subtasks* it sets up, the *control* operator it imposes on the subtasks, and the *conditions* under which this method is applicable. Figure 2.5 shows the general semantics used to describe a method used by KRITIK2[3]. The control slot contains a nested list, where tasks within the same nesting scope are performed as specified in the opening of the nest scope, that is *serial*, *parallel* or *loop*.

The control operators are the syntactic operations on the organization of the task structure. Two tasks are either serial to one another, or can be executed in parallel, or there is a loop thread around one, or a set of them. Each control operator has a name and particular information that needs to annotate one of its instantiations.

[2]This graph, representing task semantics, was taken from [Stroulia and Goel 1994].

[3]This graph, representing method semantics, was taken from [Stroulia and Goel 1994].

Tsk(task) := (name, prototype, instantiations, info-state$_{\text{input}}$, info-state$_{\text{output}}$,
semantics, by-methods||procedure, subtask-of, under-conditions)
where
name, the name of the task.
prototype := Tsk, a task that accomplishes a transformation equivalent to or more general than the transformation of the current task.
instantiations,
info-state$_{\text{input}}$:= {Info-Type}*, the input information.
info-state$_{\text{output}}$:= {Info-Type}*, the output information.
semantics,
by-methods := M$^+$, a list of methods potentially applicable to the task.
procedure, the name of the program module which accomplishes the task, i.e. whose functional abstraction the task is for the leaf tasks only.
subtask-of := Tsk, the task which this task is a subtask of.
under-conditions := p(info-state$_{\text{input}}$)*, a set of predicates on the input information of the task, under which it is meaningful to accomplish the task.

Figure 2.4: Task Semantics

M(method):= (name, under-conditions, applied-to, subtasks, control, procedure)
where
name, the name of the method.
under-conditions := p(info-state)*, a set of predicates that needs to be true in order for the method to be applicable to the task. The types of information on which these predicates are applied are all information types , the values of which have been produced before the method selection.
applied-to := Tsk, the task to which the method is applicable.
subtasks := Tsk$^+$, a set of subtasks into which the method decomposes the task it is applied to.
control := ctrl-op(subtasks(M))*, a set of control operators applied to the subtasks of the method. Control operators define a partial order among these subtasks. They define precedence among tasks, potential parallelism and repetition of tasks until a condition is met.
procedure,

Figure 2.5: Method Semantics

Domain Concepts: The types of information that KRITIK2 uses for its design task are represented at an abstract level of domain concepts. The domain concepts represented in CANAH-CHAB are DESIGN-CASE, DESIGN-FUNCTION, FAULT, DIFFERENCE, COMPONENT, SUBSTANCE, ROOT-NODE, MEMORY-NODE, PROPERTY-NODE, VALUE-NODE, PROPERTY, VALUE, etc. These are the concepts of the domain of reasoning, and as such they have *attributes*, other than their *name*. An interesting issue is that some of these concepts do exist in the world, such as COMPONENT and SUBSTANCE, while some of them do not but are rather organizational concepts imposed by the mental model used to represent the world, such as PROPERTY-NODE and VALUE-NODE. The concepts of the former kind can be discussed in a manner dependent on the domain but independent of the reasoning task that operates on them. The concepts of the latter kind require a commitment on the model and, as such, a commitment to the tasks that will make use of it. Figure 2.6 shows the general semantics of a domain concept[4].

Problem variable: Problem variables are *instances* of domain concepts used by the reasoner. A problem variable is specified in terms of its *semantic* and *syntactic* types and of the *values* it can take. Figure 2.7 shows the general semantics of a problem variable[5].

[4]This graph, representing domain concept semantics, was taken from [Stroulia and Goel 1994].
[5]This graph, representing problem variable semantics, was taken from [Stroulia and Goel 1994].

DC(domainconcept) := (name, domain, attributes, identity-test, relations, produced-by, input-to)
where

name, the name of the domain concept.

domain, the data structure with the legal values for the object; only the enumerated objects have domains, and usually this is the case for the objects that directly refer to objects in the world, as opposed to conceptual objects.

attributes := (name, function, type)*, the set of attributes characteristic of the world object. Each one of them is specified in terms of a name, a definition of a function evaluating its value given an instance of the domain concept object type, and its type which can be either another domain concept or a number.

identity-test, the definition of a function to evaluate identity in the domain of the concept.

relations := (name, table)*, the set of domain relations applicable to the domain concept. Each one of them is specified in terms of a name and an association table where the knowledge of the problem solver regarding this relation resides.

produced-by := T*, the set of tasks that can produce as output such domain concept object.

input-to := T*, the set of tasks that use such domain concept object as input.

Figure 2.6: Domain Concept Semantics

Problem-Variable := (semantic-type, syntactic-type, value)
where

semantic-type := DC, a domain concept of which this type of information is an instance.

syntactic-type := simple, multiple, specifying whether this information type consists of one or a set of domain concepts.

value, the value the problem variable can have.

Figure 2.7: Problem Variable Semantics

CHAPTER III

USE OF SBF MODELS

The use of SBF models and meta models helps the system designer make the functioning of physical devices and the reasoning processes of an autonomous system explicit to the user so that she has a better understanding of the objects and activity performed by the system. In order to make the design objects and the design processes explicit to and understandable by the user, this thesis proposes to provide the system with three necessary features: Illustration, Explanation and Navigation. Using the SBF language enables Canah-chab to possess (i) models of the devices on which Kritik2 is working and their functioning, and (ii) meta models of Kritik2's conceptual knowledge and process knowledge. How this knowledge is organized in Canah-chab and how the system was built are discussed in section 3.1. The use of SBF models to support illustration, explanation and navigation capabilities is discussed in the other sections of this chapter.

3.1 Canah-chab's Graphical Interface

The graphical interface of Canah-chab has been designed using the Garnet user interface development environment [Myers 1990]. The Garnet environment contains a set of tools to design and implement highly-interactive, graphical, direct manipulation user interfaces. It is implemented in Common Lisp and interfaces to the X window manager. It provides an extension of Common Lisp with some syntax for object-oriented programming and constraints.

The Garnet system provides a set of *widgets* which are collections of windows, buttons, menus, scroll bars, etc., that was used to create the graphical interface of CANAH-CHAB. In addition to this environment, the Garnet system provides a set of object-oriented language features. In particular, it supports a prototype-instance model for graphical objects [Lieberman 1986] that enables a dynamic, flexible definition of graphical objects and facilitates their reuse. It enables the definition of a prototype for a particular kind of object and supports the creation of as many instances of this object as needed. This feature is very useful in the design of CANAH-CHAB's interface since the same kind of knowledge is displayed many times throughout the system. For example, CANAH-CHAB contains the definitions of the prototypes of the graphical objects that represent the state of a substance, a transition between two states in the SBF model of a physical device, a task with its input, output, method and subtasks, etc. Each time it wants to display a specific state, transition, etc., it creates a specific instance of the appropriate prototype.

Interaction between CANAH-CHAB and the user is done through the use of *interactors* [Myers 1990] which are encapsulations of input device behaviors. In the current interface, the interactions are handled through the use of the mouse. However, keyboard inputs can also be handled by the system and should therefore be used in future versions of CANAH-CHAB.

The information and commands to load and run CANAH-CHAB are provided in Appendix A. A complete sequence of screens from a session of CANAH-CHAB is provided in Appendix B. It shows the illustration and explanation of the design of a flashlight circuit.

3.2 Illustration of Physical Devices and Design Processes

3.2.1 Motivation

As discussed before, an intelligent tutoring system has to provide users with access to knowledge of physical devices and their functioning, and the knowledge and reasoning processes of an expert designer. In order to understand the autonomous design system's processes, and also acquire skills from it, it is necessary for the knowledge used by the system to be presented to the user. KRITIK2 uses three different kinds of knowledge. First of all, the system has some deep knowledge about the functioning of devices. This knowledge is captured in the SBF models of physical devices and has been discussed in section 2.1. The system's understanding of how devices work needs to be presented to the user so that users can also acquire some necessary knowledge about the devices' functioning. The comprehension of existing devices will help them solve subsequent design problems, by identifying when a previous design is applicable and adapting it in order to obtain the desired function.

The second kind of knowledge that any system in general, and CANAH-CHAB in particular, has is the knowledge about the domain in which it is operating. A memory of previous design cases, a memory of the existing substances in the world along with their properties, general guidelines, libraries of the functions of components, etc., are the kinds of knowledge a user also needs to acquire in order to master the design activity. Therefore, domain knowledge also has to be presented to the user.

The third knowledge that the system possesses is about the reasoning strategies involved in the activity. CANAH-CHAB's meta models specify the tasks and subtasks to perform, and also the methods that can be applied to each task. This functional and strategic knowledge, in addition to the domain knowledge and the knowledge of the devices' functioning, need to be explicitly stated by the system and presented to the user. Reasoning strategies form an important and necessary part of the knowledge that the user needs to acquire to fully master the activity. Making the system's reasoning processes explicit to the user is a necessary step to facilitate the design skills learning.

3.2.2 Issues

The important question is how and when to present information about the devices, the domain, and the processes to the user. The amount of knowledge that the system has is large and could even be larger in the future. It is impossible to present to the user all the information at the same time. Therefore, relevant parts of the knowledge need to be provided at the appropriate times. The issue now becomes how to present parts of the information in such a way that it is accessible and comprehensible to the user. The solution we propose is to illustrate the knowledge captured by the SBF models of physical devices and the SBF meta models of KRITIK2's reasoning. Each SBF model provides enough information about a device to illustrate and explain its functioning. And the meta models of the system's domain and process knowledge capture all the relevant information to illustrate the design process, as instantiated in KRITIK2. The information captured by the SBF models and meta models of KRITIK2 is graphically presented to the users in CANAH-CHAB in the context of some *illustrative examples.*

3.2.3 Illustration Techniques

Since the information to present is of different kinds, standard techniques are used to illustrate them. First of all, both graphical and textual illustration are provided depending on the type of information. KRITIK2 only presents some information about the evolution of the artifact in a text format that was difficult to understand and follow.

CANAH-CHAB presents information about the devices and the knowledge graphically. In order to follow the consistency guideline, it uses the same graphical standards to represent the same kind of information. For example, all the behavioral states are represented by some blocks with information about the substance and the location inside, while all the behavioral transitions are specified by arrows between two states and a pointer to general information about the transition. Figure 1.5 shows an example of the illustration of the states and transitions in the causal model of an electrical circuit. CANAH-CHAB takes advantage of the object-oriented features provided in the Garnet interface development tool for these graphical objects. Another benefit of the prototype-instance model of this tool is that it enables the modification of the illustration of all the instances of a type of object such as state, transition, tasks, methods, inputs, etc., by modifying only the overall prototype of this object.

Since the domain knowledge and the process knowledge of the system are not flat, tree-like representations are used to illustrate the different levels when it is necessary. Figure 1.6 illustrates the decomposition of KRITIK2's task structure.

Highlighting techniques are also used throughout the illustration in order to show the relevant or current information the user needs to look at in a particular concept. For example, in Figure 1.6 of the design task and its subtasks, the current task to be performed by the system is highlighted using a thick-lined frame.

3.2.4 SBF Models and Illustration of Physical Devices

As it has been discussed before, CANAH-CHAB illustrates the functioning of physical devices by graphically presenting the knowledge captured in the SBF models of the physical devices known or designed by KRITIK2. Different graphical objects have been designed corresponding to the different information contained in these models. A device and its functioning are illustrated by the function of the device, its behavior(s) and its structure. To each one of these views corresponds a particular kind of "screen" that contains the relevant information. CANAH-CHAB's interface provides the user with three buttons to display the different views of the qualitative description of a physical device: Function, Behavior and Structure. The user can select one of these to display a particular view of a physical device, i.e., to display the information explaining its function, its internal causal behaviors or it structure. The structure view of the device hasn't yet been graphically implemented even though the information is available in KRITIK2's SBF models of physical devices. In Figure 3.1, representing the function of a nitric acid cooler, the buttons concerning the different views are visible on the right part of the screen.

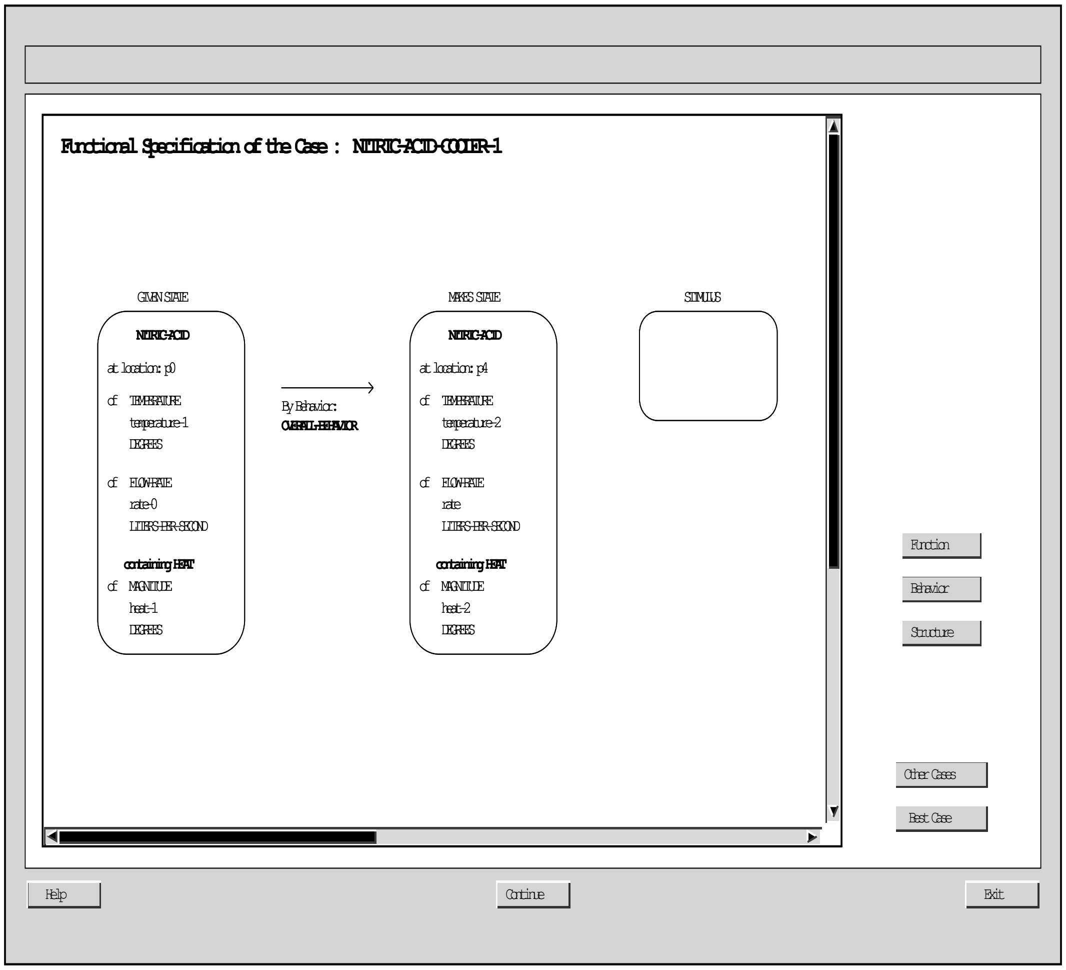


Figure 3.1: The function of a Nitric Acid Cooler

Two other buttons enable the user to display the functioning of other physical devices, whose functions are closely related to the desired function, or to display the functioning of the device that is the best match for a given design problem.

In CANAH-CHAB, the object representing a state is a *roundtangle* (rectangle with round corners) containing slots for information such as the name of the substance, the properties and values of the substance, the location (if specified) and the name and properties of the substance contained (*e.g.*, heat) if it exists. The slots of an instance of such an object are filled by accessing the information contained in the SBF models of a physical device. Two instances of this prototype are displayed in Figure 3.1 showing the function of an acid cooler. The state on the right is the one that this acid cooler device makes, given the state that is displayed on the left. This function is achieved by a particular behavior whose name is displayed between the two states. The user can either click on this name or on the "behavior" button to display the internal causal behavior that enables the achievement of this function. Figure 3.2 shows the resulting display of the overall behavior of an acid cooler.

The object representing a transition between two states is a set of arrows, a name identifying the transition and also an *interactor* that enables the user to click on the name to get more information about this transition. The graphical illustration of a transition is provided in Figure 3.2. Clicking on the name of a transition makes visible a menu that provides the user with a choice of available information concerning this transition. In the Nitric Acid's behavior shown in Figure 3.2, the result of the user clicking on the name of the transition is displayed in Figure 3.3.

In Figure 3.3, showing the behavior of the nitric acid in the nitric acid cooler example, the menu is visible on the right side of the screen. In this example, the user can ask for more information about the function used to achieve this transition, in which case a window pops up telling the user that the transition occurs because of

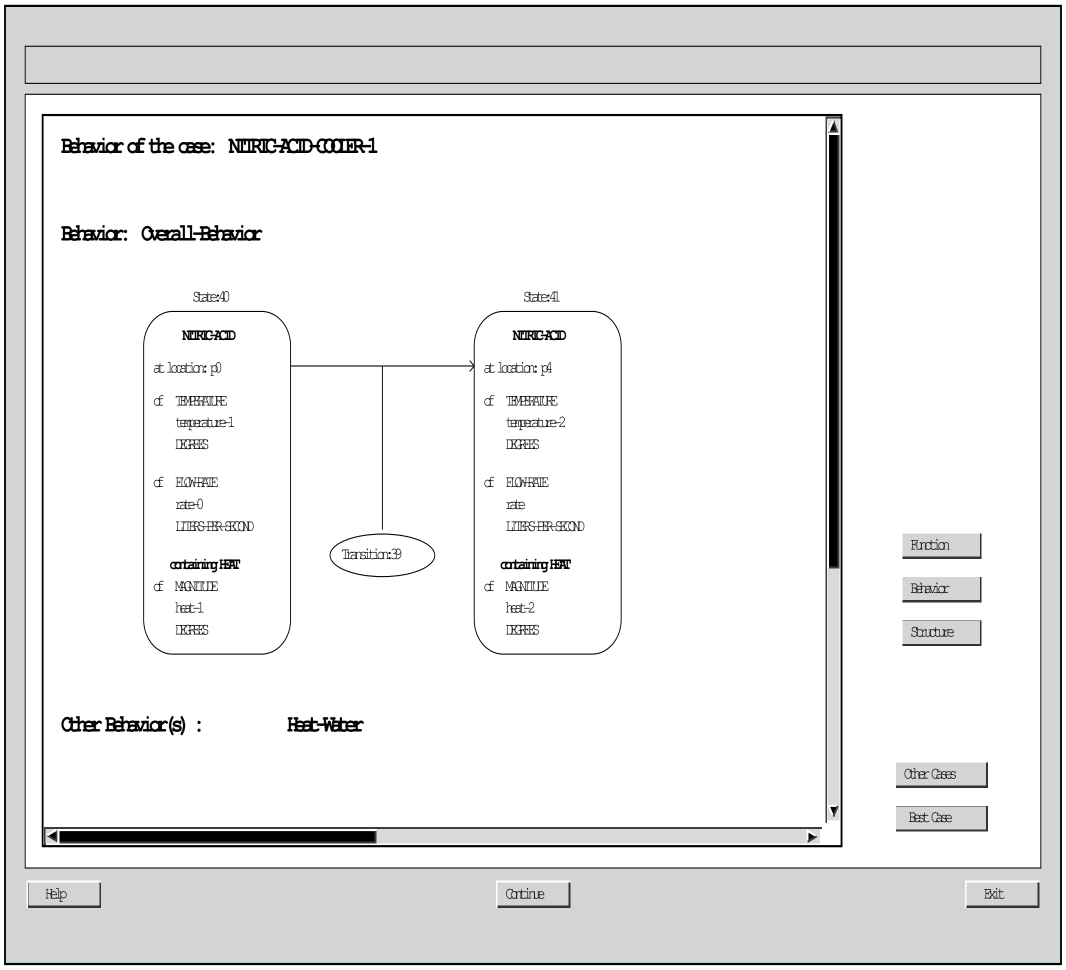


Figure 3.2: The Behavior of the Nitric Acid in a Nitric Acid Cooler

the function of a component of the device. Figure 3.4 shows this pop-up window.

Another choice by the user would have been to click on "By Behavior" in the menu. This shows that the transition about which the user is asking for more information occurs under the condition that another internal causal behavior of the physical device is achieved. Clicking on this new option would immediately lead to the display of the behavior of the water in this device. The internal causal behavior of the water in the Nitric Acid Cooler is shown in Figure 3.5.

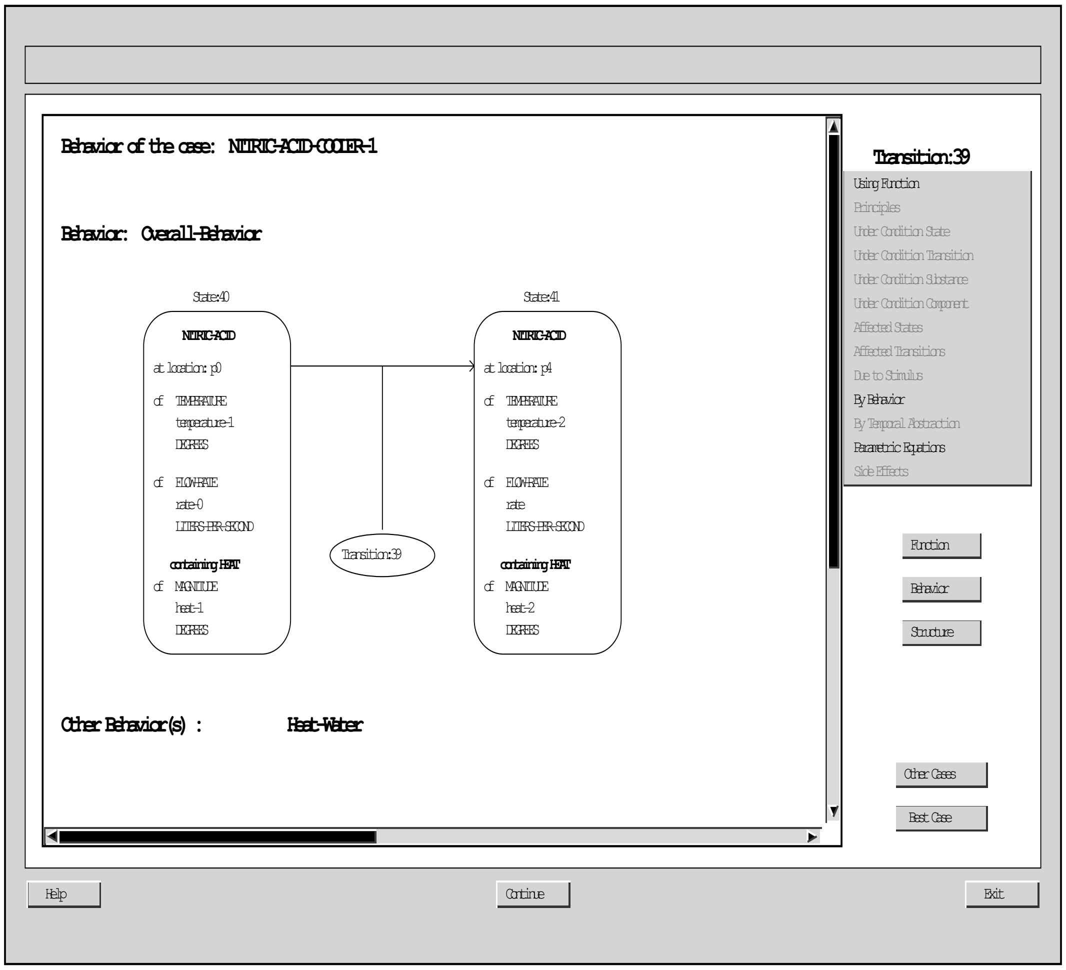
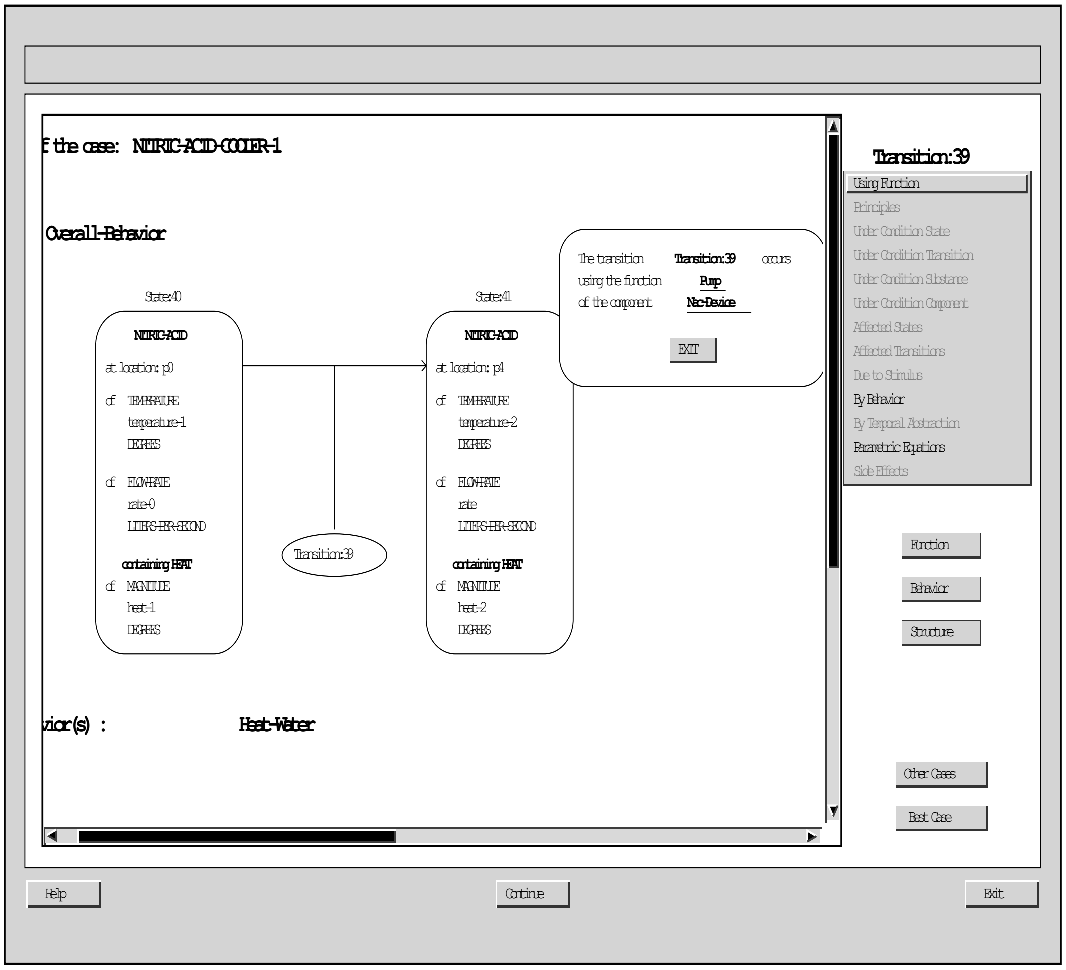


Figure 3.3: More Information about a Transition

3.2.5 <u>SBF Meta Models and Illustration of Design Processes</u>

Canah-chab illustrates the knowledge and reasoning of Kritik2 by providing graphical screens representing the information contained in SBF meta models of Kritik2's design processes.

Tasks screens: Tasks and subtasks are represented on the screen as rectangular objects. Any instance of a task or subtask object contains, in text, the name of the specific task that it represents. Input/output data types are represented by arrows



The transition **Transition:39** occurs using the function **Pump** of the component **Nac-Device**

Figure 3.4: Transition using the Function of a Component

flowing into or out of task rectangles labeled with text descriptions. Figure 3.6 shows the completed task of adaptation as a rectangle on the left of the screen.

The adaptation task is performed in KRITIK2 by using the "model-based adaptation" method. As shown in Figure 3.6, a method is represented as an oval containing the name of the specific method and is positioned to the right of the task to which it applies. Procedures are represented by their names in text.

The subtasks initiated by a method are displayed on the right of the oval representing that method. In Figure 3.6, the subtasks of the *design adaptation* task are

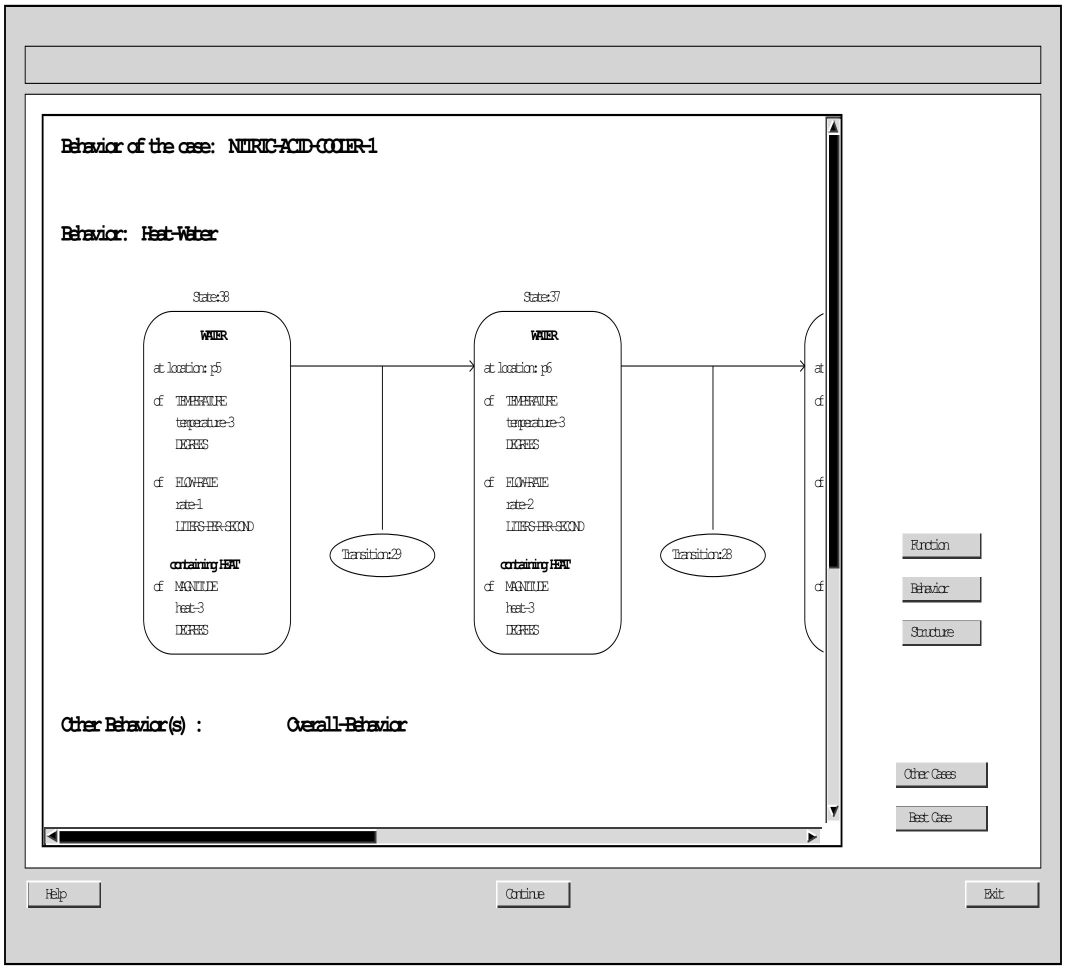

Behavior of the case: NITRIC-ACID-COOLER-1
Behavior: Heat-Water
State:38
WATER
at location: p5
of TEMPERATURE temperature-3 DEGREES
of FLOW-RATE rate-1 LITERS-PER-SECOND
containing HEAT
of MAGNITUDE heat-3 DEGREES
Transition:29
State:37
WATER
at location: p6
of TEMPERATURE temperature-3 DEGREES
of FLOW-RATE rate-2 LITERS-PER-SECOND
containing HEAT
of MAGNITUDE heat-3 DEGREES
Transition:28
Other Behavior(s) : Overall-Behavior
Function
Behavior
Structure
Other Cases
Best Case
Help
Continue
Exit

Figure 3.5: The Behavior of the Water in a Nitric Acid Cooler

presented vertically since they occur serially. The three subtasks in this example are the *computation of functional differences*, *diagnosis* and *repair*. These subtasks are represented by the same graphical objects used for the task of *design adaptation*.

Information Screens: Canah-chab not only illustrates to the user the tasks and subtasks performed by Kritik2 and the methods used in the design process, it also illustrates the knowledge used by the system as input to these tasks and the results of the transformation. Information screens are textual or graphical illustrations of the

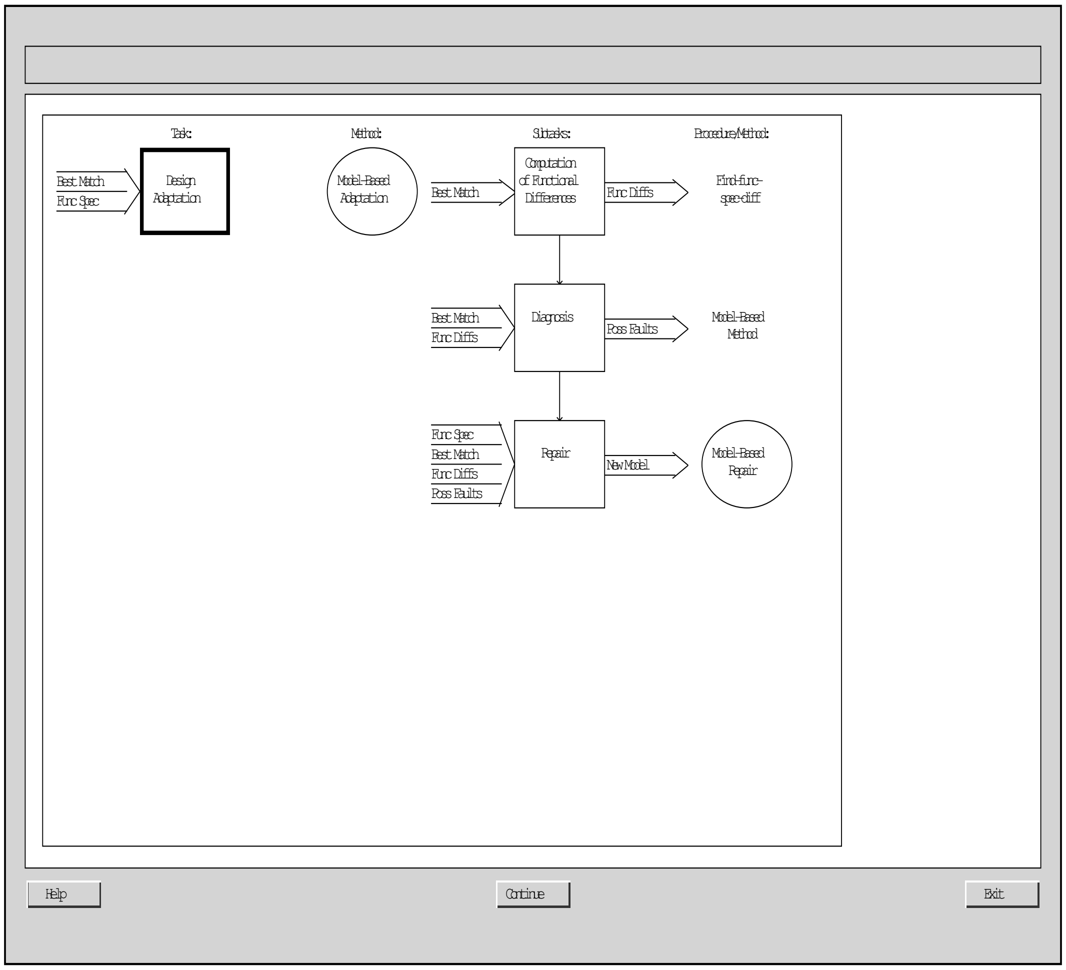
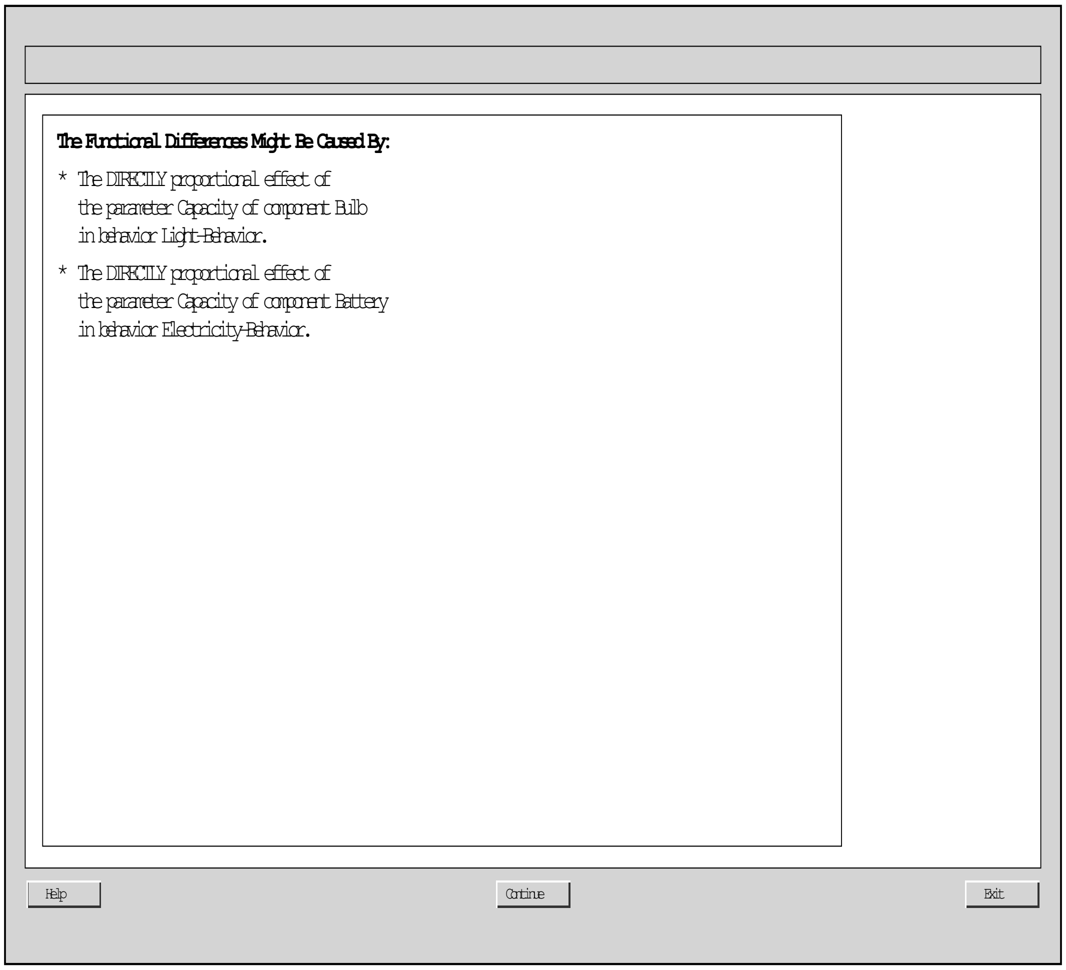


Figure 3.6: The Complete Design Adaptation Task

input and output data types used by the system. Figure 3.7 shows an information screen illustrating the result of the diagnosis task performed by KRITIK2. It presents the possibles causes of differences between the desired function of a device and the function of the device that was retrieved from the memory of devices.



The Functional Differences Might Be Caused By:

* The DIRECTLY proportional effect of
  the parameter Capacity of component Bulb
  in behavior Light-Behavior.
* The DIRECTLY proportional effect of
  the parameter Capacity of component Battery
  in behavior Electricity-Behavior.

Figure 3.7: An Information Screen: The Possible Causes of Differences

3.2.6 CANAH-CHAB's Scenario

In addition to being able to graphically show tasks and methods on the screen, CANAH-CHAB is able to show KRITIK2's design problem-solving process to users by graphically representing problem solving control structure through a task structure when tracing through illustrative examples of problem solving. Situating the design activity in real examples enables the presentation of the different kinds of knowledge used by the system in their contexts. Therefore, the knowledge about the domain and the functioning of devices is provided when it is relevant and necessary to the user in the overall problem solving example. Furthermore, it enables the user to ground the acquired knowledge in the real world.

The user is presented with a graphical illustration of the design problem solving process in the following manner. Canah-chab invokes Kritik2 to solve a given design problem from a set of pre-defined problems. Kritik2 solves the problem using a case-based reasoning method. The results from Kritik2 are interpreted and displayed by Canah-chab. Canah-chab presents the user with a sequence of task screens that show Kritik2's design process. Between task screens, it also illustrates the memories of the system, the functioning of physical devices, and other information screens, as appropriate.

The previous scenario led to the following scheme for showing the flow of problem solving in the context of illustrating a particular sample problem:

- Kritik2's design expertise can be illustrated by presenting a sequence of screens to the user.

- Screens in the sequence graphically represent task structure information or specific instances of input/output data types (instantiated in the context of the current example).

- The user displays the next screen in the sequence by clicking on the *continue* button provided at the bottom of Canah-chab's screens.

- Task screens indicate the currently active (sub)tasks by highlighting the outlines of the rectangles that represent them.

- Task screens indicate subtasks that will be executed in the future by making the outlines of their rectangles into dotted lines.

- Task screens indicate terminated subtasks by representing them with normal rectangles.
- Task screens do not show the inputs or outputs of subtasks that have not yet been executed.
- Task screens do not show the outputs of currently-active (sub)tasks.
- The states of tasks, and the visibility of their inputs and outputs, change from one task screen to the next in the screen sequence.
- KRITIK2's code was augmented so as to transfer control to CANAH-CHAB, which then invokes the appropriate screens, at the appropriate places during its execution.

Figure 1.6 and Figure 1.7, shown previously, are examples of task screens. An example of a sequence of screens shown by CANAH-CHAB in an illustrative example of KRITIK2 solving a design problem is provided in Appendix B.

3.3 Explanation of Physical Devices and Design Processes

3.3.1 Motivation

In addition to illustrating the functioning of physical devices and the design process, design and learning environments must provide explanations of the functioning of these devices and how the reasoning of an autonomous design system works. In the master-apprentice relationship, the master generally explains what he is doing and why while he is performing the activity. This phase of the relationship is very important since it forces the master to articulate his reasoning and to make explicit the knowledge he uses. It enables the apprentice to acquire concepts that are otherwise implicit and justifies the actions of the master. The environment needs to provide explanations and justifications of the knowledge it uses, the tasks and subtasks it performs, and the methods it selects.

3.3.2 Issues

The main problem in explaining the functioning of physical devices and design processes is deciding the amount of detail and information that an explanation should contain. It should be detailed enough so that the user gets a clear understanding of what it was supposed to explain, but not too detailed so that the user can understand the general idea and not get lost in low level details. Another issue that arises is deciding when to expose the user to some explanations. Should the system provide explanations all the time? Or should the user ask for explanations about something she hasn't understood or is not sure about?

3.3.3 Explanation Techniques

In order to not overload the user with explanations that might not be needed, the presentation of explanations should be optional. Nevertheless, the user should always be able to ask for an explanation at any time while using the system. Making explanations discretionary to the user might even facilitate the learning of the activity. In addition to the content of the explanation, the users must reflect about the activity and decide when more information and a complete explanation are needed.

Access to explanations should be available at all times and made easy by the use of a graphical button on the screen that gives access to a menu of explanations. It is necessary to note here that the explanation part concerns the system's reasoning and the knowledge it uses, not the commands and features of the system interface. The later are provided separately by a "help" button.

In order to provide efficient explanations, the explanation button should be context-sensitive. The explanation queries are restricted to the current part of the activity that the system and the user are looking at, but the content of the explanations might deal with other parts of the system's knowledge and reasoning, if it is necessary. Furthermore, because of the hierarchical nature of the knowledge organization and the task structure, the explanations can be of different levels. This solves the issue of the level of detail that should be provided.

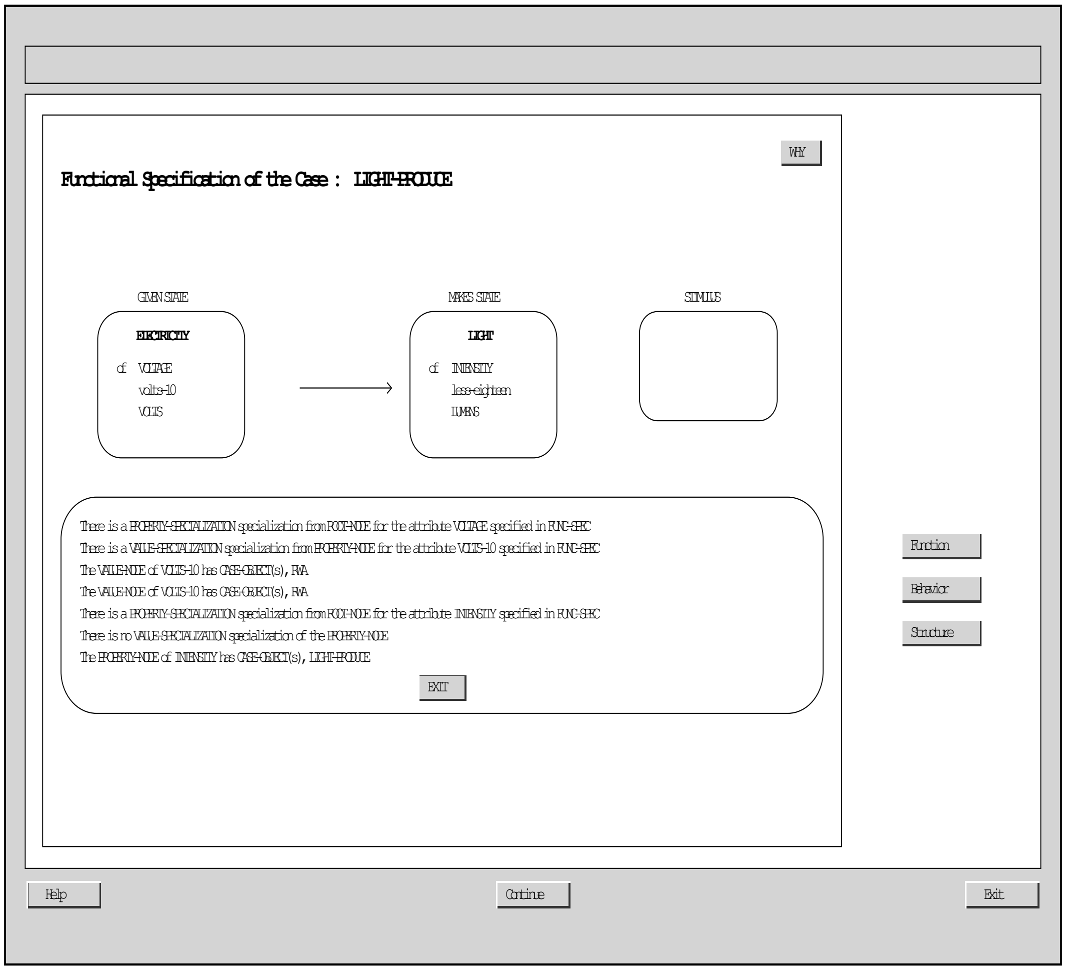
3.3.4 SBF Models and Explanation of Physical Devices

The qualitative view of devices, captured in the SBF models of physical devices, expresses an explanation of the functioning of these devices. By illustrating the SBF model of a device and making the knowledge contained in the model explicit and transparent, the user is provided with an explanation of how the device works. In CANAH-CHAB, the functioning of physical devices is explained to the user by graphically representing the content of SBF models. How the system works is explained by providing the internal causal processes or behaviors of a device. Figures 3.1 through 3.5, shown before, provide an example of the explanation of the functioning of a nitric acid cooler.

3.3.5 SBF Meta Models and Explanation of Design Processes

The advantage of using SBF meta models of the design processes is that the system can perform some simulation. It, therefore, allows the system to generate explicit explanation about the reasoning processes while other systems can only provide the user with some *previously generated* explanations. In systems of the latter kind, explanations about the knowledge and the task are pre-recorded by the designers of the system and concern only the actual behavior of the system. On the other hand, when an intelligent tutoring system such as CANAH-CHAB has a meta model of an autonomous design system's processes, it can generate the explanations itself.

Using the knowledge captured in the SBF meta models of KRITIK2's design process, Canah-chab can generate justifications and explanations of expected behaviors. It can explain a task, justify the use of a method, simulate the result of the process given a particular specification and justify some results. Figure 3.8 shows the justification for the retrieval of two cases from Kritik2's memory of past device designs given a functional specification.



Functional Specification of the Case : LIGHT-PRODUCE

There is a PROPERTY-SPECIALIZATION specialization from ROOT-NODE for the attribute VOLTAGE specified in FUNC-SPEC
There is a VALUE-SPECIALIZATION specialization from PROPERTY-NODE for the attribute VOLTS-10 specified in FUNC-SPEC
The VALUE-NODE of VOLTS-10 has CASE-OBJECT(s), RWA
The VALUE-NODE of VOLTS-10 has CASE-OBJECT(s), RWA
There is a PROPERTY-SPECIALIZATION specialization from ROOT-NODE for the attribute INTENSITY specified in FUNC-SPEC
There is no VALUE-SPECIALIZATION specialization of the PROPERTY-NODE
The PROPERTY-NODE of INTENSITY has CASE-OBJECT(s), LIGHT-PRODUCE

Figure 3.8: Justification of the Result of the Retrieval Task

3.4 Navigation through Physical Devices and Design Processes

3.4.1 Motivation

In addition to the illustration of its domain and process knowledge, a system must provide navigation through them to the user. Navigation can not only provide flexibility, it can also support the exploration of the system by the user, therefore giving her the initiative to learn more about the system's memory of past design cases, its conceptual knowledge and its processing knowledge.

Allowing the user to browse through the system's knowledge enables her to see parts of the knowledge that maybe weren't directly provided by the illustrative examples. Both the content and the form of the information presented are different. Navigation enables the user to have a more complete view of one kind of knowledge. For example, browsing through the case memory provides the user with a general presentation of all the design cases in memory. An illustration of the design problem-solving activity would have only presented the user with the best matching case, and would have situated it in the context of the retrieval task with other forms of knowledge, such as knowledge about substances, components' function, etc.

3.4.2 Issues

Two issues arise when talking about navigation through the system's knowledge. The first issue, which was also a problem for illustration, is that the knowledge of the system can be large. What are the navigation techniques or guidelines to make browsing through complex spaces efficient? The second issue is that users tend to *get lost* when they navigate through large systems. Therefore, we need to provide hints to the user to allow her to know "where she is" in the system's knowledge.

3.4.3 Navigation Techniques

In order to avoid having the users get lost while browsing through the system, standard techniques suggest providing context to the user. Providing general graphical context associated with visual clues to situate the current position enables the user to visualize where she is navigating, what kind of organization she is browsing through and what kind of information she is looking at, etc. Figure 1.7 shows how the context for the retrieval task is provided before going deeper inside the browsing of the task in Figure 1.8. It shows the ordering and hierarchy of the tasks, combined with other information.

To support effective browsing in large complex spaces, standard techniques also suggest using several levels of display. For example, the retrieval task is displayed in two different windows, so that it shows the hierarchy of the information displayed. In order to support faster navigation through complex hierarchical displays, the system also provides direct access facilities such as in the case memory.

3.4.4 Navigation in the SBF Model of a Physical Device

To provide the user with a better understanding of the functioning of physical devices, Canah-chab supports navigation through the SBF models of devices. The user can navigate between the different Structure, Behavior, and Function views of a device in order to get a more general understanding of its functioning. Figures 3.1 through 3.5 show the different views of a nitric cooler and how the user can navigate between these illustrations.

3.4.5 Navigation in the System's Memory

Using SBF meta models of the system's knowledge and reasoning enables the user to have access to the complete knowledge base used by the autonomous design system. The user can, for example, browse through the memory of past design cases, the memory of substances, physics principles, etc. The advantage of using SBF meta models of the design processes is that the browsing can be guided by the system. In general, browsing is an activity initiated by the user and completely unguided by the system. But with meta models, the system can perform some simulation and restrict the browsing space depending on some user/system defined conditions. For example, given a particular design problem, the system can restrict the memory of design cases to the cases that are relevant to the problem, i.e., cases that could be adapted to produce the desired solution.

CHAPTER IV

DISCUSSION

4.1 Results and Contributions

This thesis describes an enabling technology to support the illustration of the functioning of physical devices and of autonomous design systems. Canah-chab, the product of this research, also supports explanation of how physical devices and design processes work. Furthermore, it supports navigation through the memory of Kritik2.

This research has shown that (i) SBF models of physical devices enable the illustration and explanation of how physical devices work; (ii) SBF meta models of an autonomous design system enable the illustration and explanation of the design process, as instantiated in the system, and (iii) the use of SBF models and SBF meta models for the two abilities indicates that the same kind of content theory, representation scheme and presentation mechanisms may be sufficient for the two abilities.

4.2 Limitations and Critiques

The current version of Canah-chab suffers from several limitations. First, the structure of a physical device, defined by its components and their relations are not yet graphically presented in Canah-chab. Another observation concerning Canah-chab is that it is maybe too faithful to Kritik2 concerning the description of the design process. Kritik2's view of design as information-transformation processes is somewhat limiting, other methods should, therefore, be proposed to support the design task. In the version of KRITIK2 used for CANAH-CHAB, values are only textual symbols for which KRITIK2 has an ordering. A more recent version of KRITIK2 deals with numerical values instead of symbols. The version of KRITIK2 used by CANAH-CHAB needs to be updated to accept these numerical values.

Second, CANAH-CHAB currently allows only limited user interaction. For example, in the current version of the system, the user can only move forward between the screens illustrating the design process. Providing backward navigation capabilities would enable the user to re-display information or trace backward through the reasoning of the system. Enabling the user to select the task or method to illustrate would provide even more flexibility in the use of CANAH-CHAB. Another limitation of the interaction capabilities of CANAH-CHAB is that, in its current stage of development, it does not let the user make any choice concerning the reasoning processes. The user can't suggest the use of a particular case to solve a problem or can't choose to apply a particular method to solve a task.

Third, the interface of CANAH-CHAB needs considerable improvement to be usable. For example, some terms such as "state:29" and "transition:38" that CANAH-CHAB displays would be completely meaningless to a potential user and need to be modified. This limitation of CANAH-CHAB's interface is due to the fact that SBF models of devices, as they are implemented in KRITIK2, contain information describing the functioning of devices in terms that are useful and meaningful for KRITIK2's task of design problem solving but were not meant to be displayed to the user.

Another critique about CANAH-CHAB's interface concerns the fact that all the information is provided in a static, *boring* manner. Adding animation to the system would better illustrate the flow of information in the reasoning processes and the flow of substances in the physical devices.

4.3 Related Work

This work connects with several lines of research in Artificial Intelligence, design, and interactive systems such as qualitative models of physical devices, meta models of problem solvers, the similarity between these two kinds of models, illustration and explanation of physical devices, illustration and explanation of reasoning processes, and case-based design.

4.3.1 SBF Models of Physical Devices

SBF models are qualitative models that capture an understanding of the functioning of physical devices at three levels: the function of a device, the internal causal behaviors by which this function is achieved, and the structure of the device enabling these behaviors. Vasandani and Govindaraj [1994] have previously used SBF models in a computer-based instructional system, called TURBINIA-VYASA, that trains operators to troubleshoot and diagnose faults in marine power plants. The simulator, TURBINIA, is based on a hierarchical representation of subsystems, components, and primitives and uses the *qualitative approximation* scheme described in [Govindaraj 1987]. At the highest level, the system knowledge is represented by *schematics*, *functional subsystems* and *fluid paths*. The lowest level of description of the system knowledge is at the component level and has three attributes: structure, function and behavior. In TURBINIA-VYASA, a component's structure refers to its connections to other components, the fluids carried by it, the gauges that are attached to it, and its association to a schematic or a functional system. The functional knowledge about a component is its intended use in the system and its contribution to higher level functions. Knowledge of a component's behavior concerns its states. In TURBINIA-VYASA, the structural knowledge and the behavioral knowledge are different under the normal and failed modes of the system.

The SBF models of physical devices used in KRITIK2 evolve from earlier models: *functional representation* ([Sembugamoorthy and Chandrasekaran 1986]) and *consolidation* (Bylander and Chandrasekaran 1986], [Bylander 1991]). Within KRITIK and KRITIK2, SBF models have been used for a variety of tasks such as case retrieval [Goal 1992], design adaptation [Goel 1991a, 1991b], design verification and redesign [Goel and Prabhakar 1991], index learning [Bhatta and Goel 1992, 1993b], and analogical design [Bhatta, Goel and Prabhakar 1994].

4.3.2 SBF Meta Models of Problem Solvers

In earlier work, Allemang [1990] has viewed a program code as a device and represented it in the functional representation language. Similarly, Weintraub [1991] has viewed a diagnostic problem solver as a device and represented its functioning in the functional representation language. Recently, Johnson [1993] has modeled how SOAR ([Laird, Rosenblum and Newell 1986]) solves problems in the functional representation scheme.

In the AUTOGNOSTIC system, Stroulia and Goel describe an approach in which a path-planning problem solver for pedestrians, called ROUTER, is viewed as an abstract device and its functioning is described using the SBF language. In AUTOGNOSTIC, the SBF meta model is used to support problem solving, monitoring and blame assignment. In CANAH-CHAB, the same SBF language is used to capture the knowledge and reasoning processes of an autonomous design system, KRITIK2. In this work, the SBF meta models are used to support the illustration and explanation of the design processes.

4.3.3 Similarity Between Models of Devices and Systems

In the context of knowledge acquisition, Davis discussed the concept of *meta-knowledge* [Davis 1979]. Davis developed a system, called TEIRESIAS, that provides an "advice-taking" interface to MYCIN [Shortliffe 1976], an expert system that diagnoses and prescribes treatment for infectious diseases. TEIRESIAS operates by working on sample problems given by an expert. The expert who disagrees with its conclusion is invited to follow the chain of reasoning that led to that conclusion until the point where a conclusion was reached when it shouldn't have or was not reached when it should have been. The expert provides new knowledge to the system to correct the mistake. In TEIRESIAS, knowledge acquisition is enabled because the program can "know what it knows". TEIRESIAS can not only use its knowledge directly, but it is also able to examine it, abstract it, and direct its manipulation. In TEIRESIAS, the *object-level* representations, describing the external world, are expressed in terms of rules. The *meta level* representations, describing the internal world of representations of the system, are also captured in terms of rules. It is interesting to note that the same kind of representations are used at both level.

In his work on meta-planning, Wilensky [1984] has shown the similarity between the knowledge about the world and the knowledge about the planning process itself. In the planner, the knowledge about the world is expressed in terms of goals and plans. Wilensky has shown that knowledge about the planning process can itself be formulated in terms of higher-level goals and plans called *meta-goals* and *meta-plans*. He shows that meta-goals and plans can be used by the same understanding and planning mechanisms that process ordinary goals and plans.

The same stance has been adopted in CANAH-CHAB. The same SBF language is used for describing knowledge about physical devices and knowledge about the reasoning processes that operate on these physical devices. Both SBF models of physical devices and SBF meta models of an autonomous design system are used in CANAH-CHAB to support uniformity of presentation and the similarity of processing. SBF models and meta models are used for illustration, explanation and navigation.

4.3.4 Illustration and Explanation of Devices

The illustration and explanation of physical devices is a classical issue in intelligent tutoring systems. SOPHIE, designed to teach troubleshooting in electrical circuits, was perhaps one of the first intelligent tutoring systems in an engineering domain ([Brown, Burton and de Kleer 1982], [Brown and Burton 1986]). The work on SOPHIE motivated much research on qualitative reasoning. Forbus [1984], for example, developed a *qualitative process theory* to describe the behavior of physical processes in terms of time intervals over which things happen or remain true. His work focuses on predicting what will happen in the next time interval and on explaining what has already happened. As mentioned earlier, our approach to the illustration and explanation of physical devices is with SBF models. SBF models differ from Forbus' view in that the qualitative process theory is general and derivational while the SBF models used in CANAH-CHAB are specific and compiled.

ASK HOW IT WORKS [Kedar *et al.* 1993] is a prototype of an intelligent interactive manual for devices. It is based on an intelligent training system, called ASK Systems [Schank 1991], that trains people by providing dialogs with experts. Instead of modeling students, the device and the reasoning process, ASK indexes information by answers and follow-up questions. ASK HOW IT WORKS builds on DEDAL [Baudin *et al.* 1993], a hypermedia system that conceptually indexes documents on mechanical devices, and on an ASK system's ability to navigate the indexed material through trainees' questions.

4.3.5 Illustration and Explanation of Processes

In GUIDON, Clancey [1987] built a tutor on top of MYCIN [Shortliffe 1976], a rule-based system that diagnoses and prescribes treatment for infectious diseases. Clancey identifies the different kinds of knowledge that must be made available for a tutor to function effectively: the performance expertise, that is augmented with support information and metalevel abstractions, and the pedagogical expertise, that is explicitly represented in terms of rules which refer to general representation scheme but not the contents of the domain. The problem with the rule-based implementation of the tutorial module is that GUIDON has difficulty following students and students have problems in acquiring MYCIN's expertise. The reason for this is that MYCIN's rules represent *compiled expertise* which is obscure to the students. In order to remedy GUIDON's shortcomings, Clancey and his team built a new version of MYCIN, called NEOMYCYN. The novelty in NEOMYCIN is that the control structure is a set of domain-independent set of metarules which explicitly represent a hierarchically organized *reasoning strategy* for medical diagnosis. The domain-independent mechanisms of NEOMYCIN are captured in a *heuristic classification system*, called HERACLES. It includes a reasoning strategy, expressed as metarules and tasks, and a language of relations between objects. The work on NEOMYCIN and HERACLES led to a new explanatory interface, GUIDON-WATCH [Richer and Clancey 1985]. In this new interface, the explanation of the reasoning process was expressed in the language of tasks. In CANAH-CHAB, the tasks and subtasks structures are similar to those in Clancey's work. However, the formulation in CANAH-CHAB is closer to Chandrasekaran's work on task structure decomposition [Chandrasekaran 1990] and on explanation using the task structure decomposition [Chandrasekaran, Tanner and Josephson 1989].

In another line of this research, the emphasis is on the visualization of the design processes. In GIL [Merrill *et al.* 1992], an interactive learning environment that supports students' problem solving as they learn LISP programming, the students can make their reasoning explicit. GIL provides graphical representations of programming to illustrate the students' reasoning process. In this graphical tutor for LISP programming, network structures that depict the flow of function calls are used to visually represent LISP programs. These structures are used to help learners visualize the abstract concepts associated with functional programming, thus facilitating their learning of this non-traditional programming paradigm. One difference between GIL and CANAH-CHAB is the granularity of the objects represented in the nodes of network structures. In GIL these nodes represent individual LISP instructions, whereas in CANAH-CHAB they are generic reasoning tasks, abstracted away from any particular implementation (even though KRITIK2 *does* provide a specific implementation of them). In addition, GIL's graphical representations are used to guide learners during the solution of specific programming problems by letting users compose networks of LISP instructions by selecting them from menus, clicking on them to ask for help, etc. In the current version of CANAH-CHAB, on the other hand, the graphical representations cannot be manipulated by the users, and serve more as didactic facilitators for the system's explanations and illustrations of problem solving.

4.3.6 Case-Based Design

Kolodner [1991] has advocated the use of libraries of past design cases to support problem solving. ARCHIE [Pearce *et al.* 1992] provides architects with a library of past design cases in the context of conceptual design of new offices building. It supports design generation and design critiquing. ASKJEF ([Barber *et al.* 1992]) is a case-based design system that, in addition to a library of past designs, provides access to design guidelines and principles in the domain of interface design. ARCHIETUTOR [Goel *et al.* 1993] is a new version of ARCHIE that takes into account the lessons learned form ASKJEF about providing access to domain knowledge. ARCHIE2 [Domeshek and Kolodner 1991] is a newer version of ARCHIE whose emphasis is on ensuring that the important lessons from each past design are communicated to the users.

CANAH-CHAB can also be seen as a case-based design system. It provides access to an external memory of past design cases. However, important lessons learned from

AskJef and ArchieTutor have been taken into consideration. For instance, in addition to information about design objects and the domain knowledge, the intelligent tutoring system also illustrates and explains the design processes. Canah-chab is an attempt to provide both kinds of information.

4.4 Future Research

4.4.1 Scaffolding

With some modifications, the Canah-chab system could also be used as a scaffolding tool, by helping the user and guiding her while she is trying to solve some problems. The SBF meta models of the design process make it possible for Canah-chab to provide advice to the user. In the current version of Canah-chab, the user observes Kritik2 solving a design problem but does not interact with it. A different version had been implemented earlier, but is still incomplete, where the problem specification was provided by the user to Canah-chab who transmitted it to Kritik2 in order to solve the given problem. This means that the illustrative and explanatory purpose of Canah-chab could be modified to serve as a design tool showing the processes that led to a design. Furthermore, instead of invoking Kritik2 at each step of the design, Canah-chab could prompt the user in the design task, invoking Kritik2's SBF meta models when the user needs support. As mentioned earlier in the explanation section, the system can simulate a problem and generate the solution with explanations. Canah-chab could, therefore, guide the user through the different steps of the task, prompting for methods and simulating the result of a task when the user doesn't know what to do.

4.4.2 Interface Development

The primary goal of this thesis was to provide an enabling technology to support the illustration and explanation of both physical devices and design processes. Thus, interface development in CANAH-CHAB has not received much attention and it is not clear whether the system's current interface is usable. CANAH-CHAB wouldn't be meaningful to a user at its current stage of development. Future work on CANAH-CHAB should lead to a better interface, first by providing terms that would be more meaningful to the potential users than the ones that are currently captured in the SBF models of physical devices in KRITIK2.

Future work on CANAH-CHAB also needs to support a wider range of user interactions. Interaction facilities are available in the Garnet environment so future versions of CANAH-CHAB should include more navigating abilities for the user. In particular, at the level of the illustration of the tasks and methods, the user should be able to go backward in the sequence of screens and even to select the tasks or methods that she wants to see illustrated. These features would provide more flexibility in the use of CANAH-CHAB.

One of the benefits of using meta models to illustrate the system's knowledge and reasoning processes is that it provides the ability for user-controlled illustrations. CANAH-CHAB expresses KRITIK2's knowledge, the organization of its memories, the ordering of the tasks to perform, and the methods applicable in a language independent of their representations. Therefore, this knowledge could be illustrated in different formats, i.e., textually, graphically, via graphs, trees or tables, depending on the user's choice.

Even though CANAH-CHAB provides some navigational capabilities, the information illustrated by the system is presented in a rather "static" way. Looking at the nature of the information displayed, it could be interesting to add *software visualiza-*

tion features to the system ([Stasko and Patterson 1992]). Indeed, some animation programs could be used to illustrate the reasoning processes, viewed as information flowing through subtasks, and the physical devices, described with some substances flowing through components. Using a software visualization system, such as TANGO [Stasko 1990], could enable a smooth animation of the functioning of the physical devices and of the reasoning processes, instead of the sequence of screens that CANAH-CHAB's interface is made of in the current version.

4.4.3 Evaluation

While CANAH-CHAB's interface is being improved, there should also be research on characterizing the category of users this intelligent tutoring system is intended for. The emphasis of this thesis was on the enabling technology to support the illustration, explanation and navigation of physical devices and design process. However the cognitive side of this research should also be addressed in the future. Describing the knowledge of the target users would also be very helpful in developing the "right" interface for them.

Once the interface of CANAH-CHAB has been improved, it should be tested by a sample population of the potential users in order to evaluate the understandability of the illustrations and explanations, the usefulness of the navigation capabilities and also the nature of what users learn from CANAH-CHAB.

4.4.4 Articulation of the User's Reasoning Activities

Using a language such as the SBF language not only enables the explicit representation of the system's knowledge and reasoning, it also provides a target language for the user to articulate her own reasoning and actions. The system exposes the user to a formal language, and gives her a vocabulary with which she can formulate her reasoning and communicate with others about the activity.

Furthermore, it might potentially enable the user's reflection on this articulation of the reasoning process. In the context of design, Schöen [1987] has argued that reflection is a fundamental constituent of design problem solving and learning. Reflection can help a student become a more skilled designer, one who produces better quality designs because she can recognize and avoid potential errors of reasoning, and one who produces designs more efficiently because she understands the interdependencies among design decisions, the applicability conditions of problem-solving methods, and the potential sequencing of the subtasks they set up. This issue of learning through reflection is critical if we are to create an environment that not only supports designers in creating new designs but also enables them to become better designers.

Providing a language to represent the system's reasoning enables users to make use of this vocabulary to explicitly and formally articulate their own reasoning concerning the processes and knowledge involved in the activity of design. Articulating their reasoning using this language can help them identify their mistakes. It can also give rise to a generalization of the activity by abstracting the reasoning away from the provided context. The use of models of the design processes and objects can therefore support reflection. Supporting these articulation capabilities is also a future direction of research for CANAH-CHAB.

APPENDIX A

CANAH-CHAB: **USER MANUAL**

This section discusses how to load and run CANAH-CHAB. Instructions are included for running several demonstrations.

CANAH-CHAB's **Directory:** The files for KRITIK2 are available in the **ktutor/Adaptation**, **ktutor/Examples**, **ktutor/Memory** and **ktutor/System** directories. The files concerning CANAH-CHAB are in the **ktutor/Tutor/demo** directory, and all the following commands need to be typed from this directory.

Loading the Garnet Software: To load the Garnet software, type the following: **/usr/local/bin/garnet**; it will automatically load many Garnet files and also the Common LISP interpreter.

Loading Garnet Objects: Once the default Garnet stuff has been loaded, load the additional Garnet items that are needed for CANAH-CHAB (e.g., code for handling mouse clicks and for displaying different types of graphical objects), by typing: **(load "kt-init1A")**.

Setting the Package: Set the active package in the LISP environment to KRITIK2 and CANAH-CHAB's by typing: **(setf *package* (find-package 'ka))**.

Loading CANAH-CHAB: Load the graphical file loader, which also automatically loads KRITIK2's code (modified to communicate with CANAH-CHAB), by typing: **(load "gdemo")**.

Starting a demo: To start the graphical illustration and explanation by CANAH-CHAB of the design of a physical device, type **(new-cbr** *problem-name***)**, where *problem-name* identifies a particular problem. Among the examples that are currently available, **problem5** illustrates the design of a flashlight circuit, **problem1** illustrates the design of a nitric-acid cooler, and **problemc** illustrates the design of a buzzer circuit.

For example, type **(new-cbr problem5)** to run the illustration of the design of the flashlight circuit. The screens displayed by CANAH-CHAB during this example are shown in Appendix B.

Starting a new demo: To start a new illustration, type **(do-cleanup)** and it will reset the system, then type **(new-cbr** *problem-name***)**, as it was described above.

Quitting CANAH-CHAB: First, double-click on the *Exit* button at the bottom of the screen to end any graphical demonstration. To end CANAH-CHAB's session and quit the Garnet environment, type **(quit)** and the operating system prompt will reappear.

APPENDIX B

CANAH-CHAB: DESIGN OF A FLASHLIGHT CIRCUIT

This section provides a sequence of screens from a session of CANAH-CHAB illustrating and explaining the design of a flashlight circuit. The commands to run this session were provided in Appendix A. This session corresponds to the flashlight circuit problem, called *problem5* in Appendix A.

The user interacts with the system by clicking on the screens with the mouse. At the bottom of each screen, three buttons are visible. The *Help* button, on the left, is meant to provide hints to the user on the functioning of CANAH-CHAB and its interface. This button is not active yet. The *Exit* button, on the right, enables the user to close the display window, quit the session and go back to the Garnet prompt. The *Continue* button, in the middle, enables the user to go on to the next screen of CANAH-CHAB.

Figure B.1 shows the first screen that is presented to the user during a session of Canah-chab. It shows the top level design task screen, i.e., the general decomposition of the *Design* task, as it is going to be performed by Kritik2. The user clicks on the *Continue* button to display the input to the *Design* task.

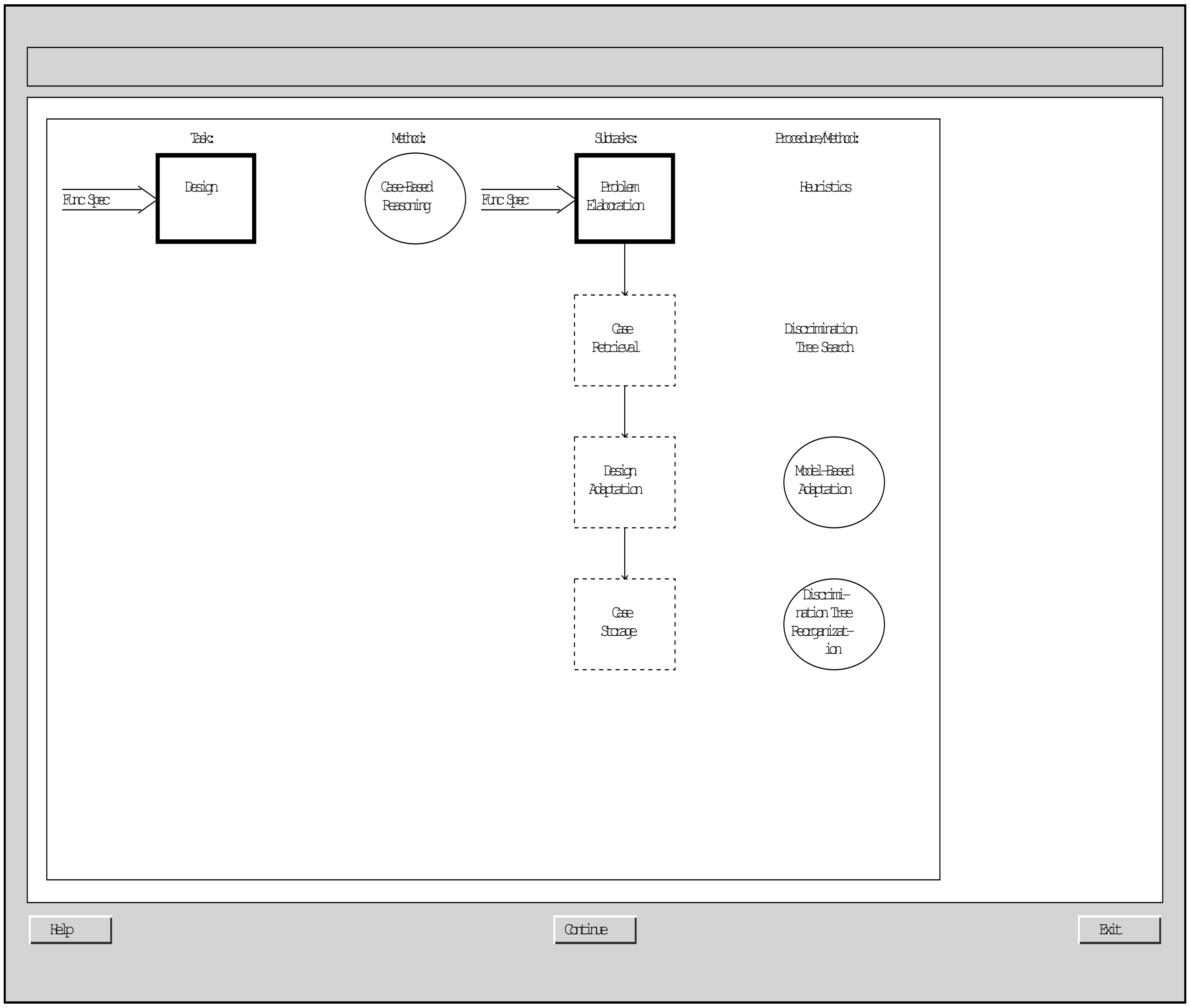


Figure B.1: The Top Level Task Screen: the Design Task

Figure B.2 shows the problem that is going to be solve by KRITIK2. It represents the functional specification of a physical device, in this case, a flashlight circuit. The user clicks on the *Continue* button to go on with the design problem solving.

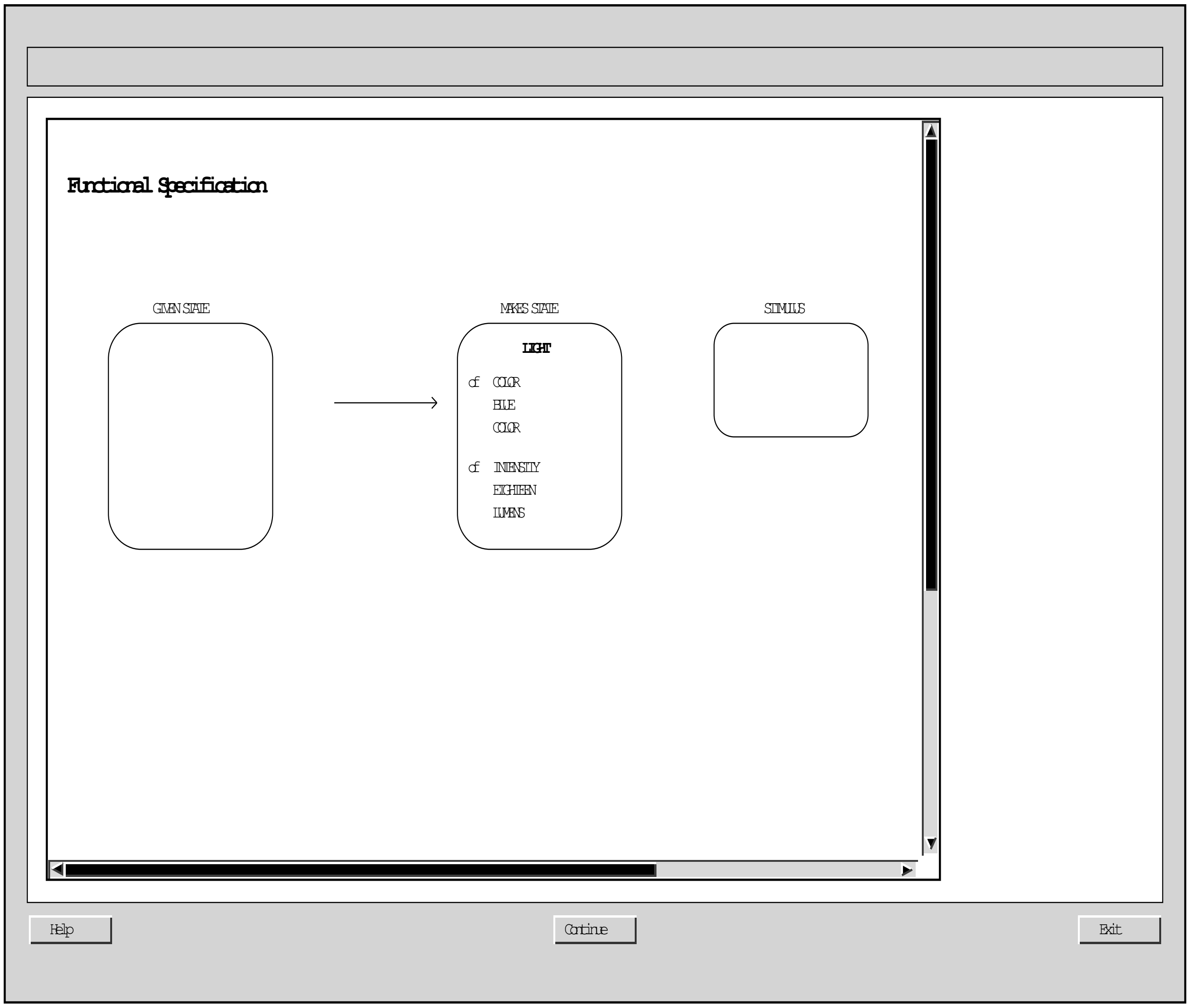


Figure B.2: Functional Specification

Once the user has clicked on the *Continue* button, the top level task screen is re-displayed. The output of the *problem Elaboration* subtask consists of some probes that are going to be used to retrieve a case and appears on the right of the subtask in Figure B.3. The user clicks on the *Continue* button to see the illustration of these probes.

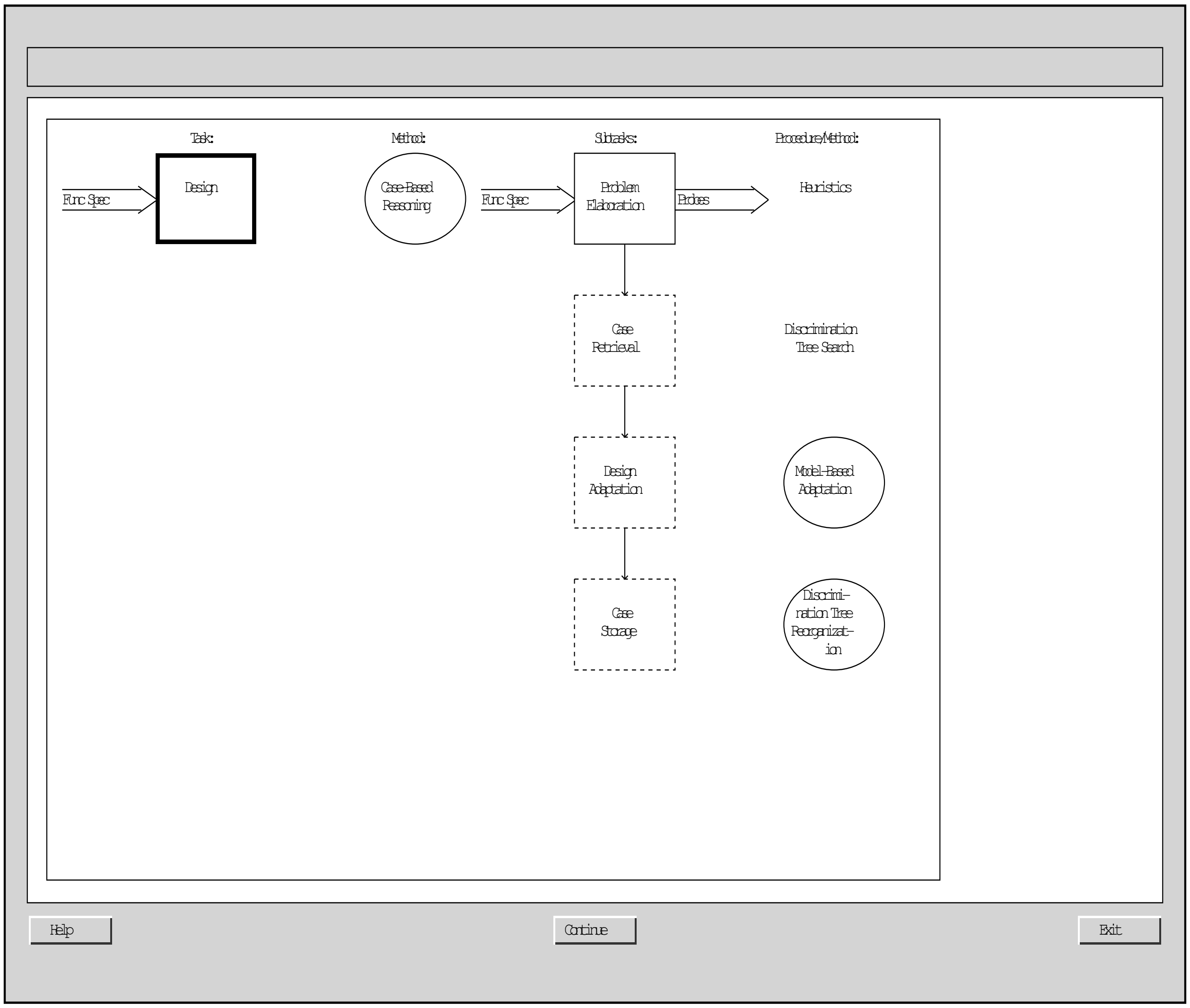
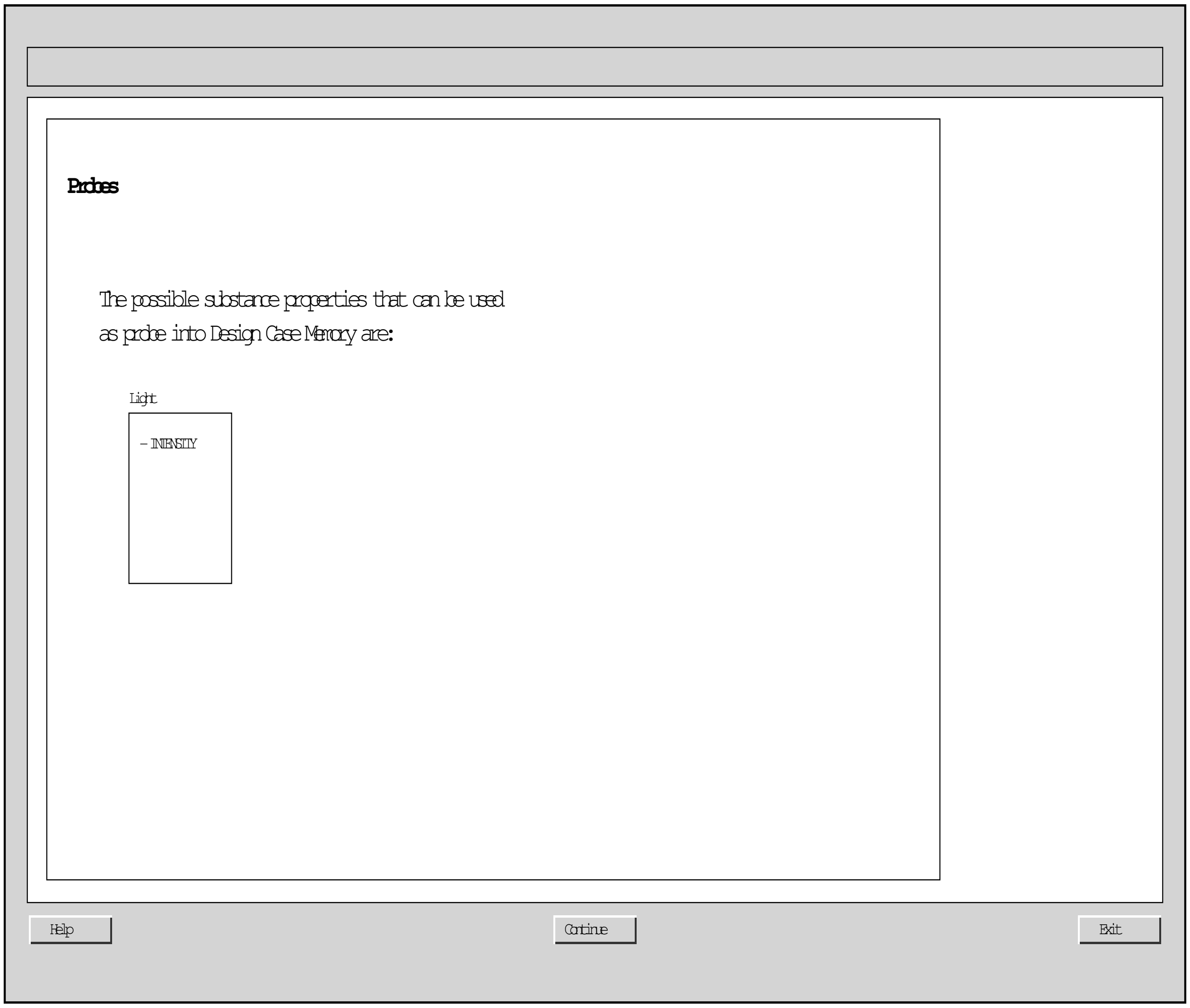


Figure B.3: Design Task Screen: After Problem Elaboration

Figure B.4 shows that KRITIK2 is going to use the property *intensity* of the substance light, specified in the functional specification of the problem, as probe in its memory of cases. The user clicks on the *Continue* button to display the next subtask of the *Design* task.


Probes

The possible substance properties that can be used as probe into Design Case Memory are:

Light

- INTENSITY

Figure B.4: Output of the Problem Elaboration: Probes

Figure B.5 shows the top level task screen again. It shows that the probes displayed in Figure B.4 are going to be used as input for the *Case Retrieval* subtask. The user clicks on the *Continue* button to to display the result of the *Case Retrieval* task.

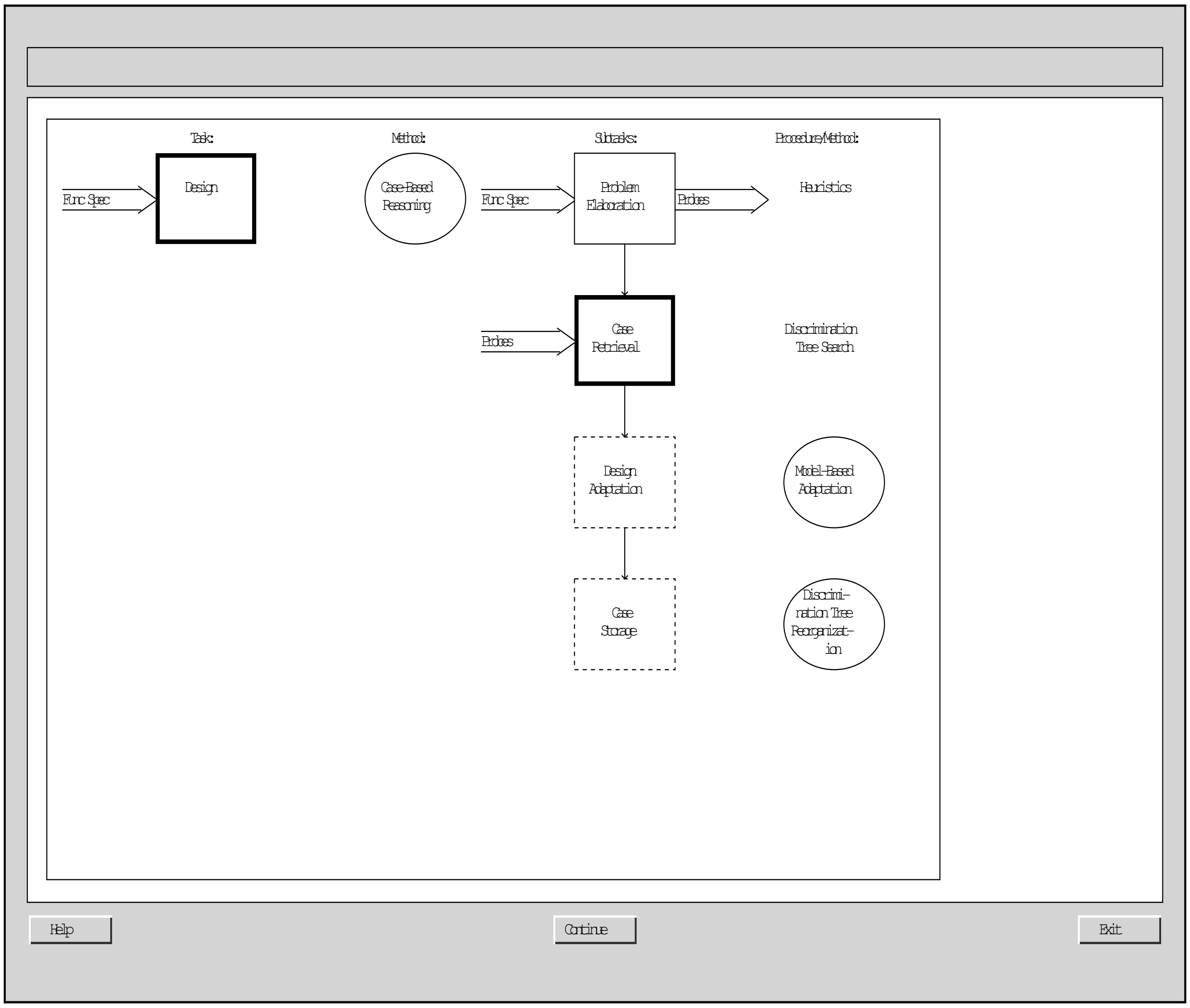


Figure B.5: Design Task Screen: Before Case Retrieval

Figure B.6 shows that the output of the *Case Retrieval* subtask is the best matching design case found in the memory of cases using the probes. The user clicks on the *Continue* button to display KRITIK2's memory of design cases.

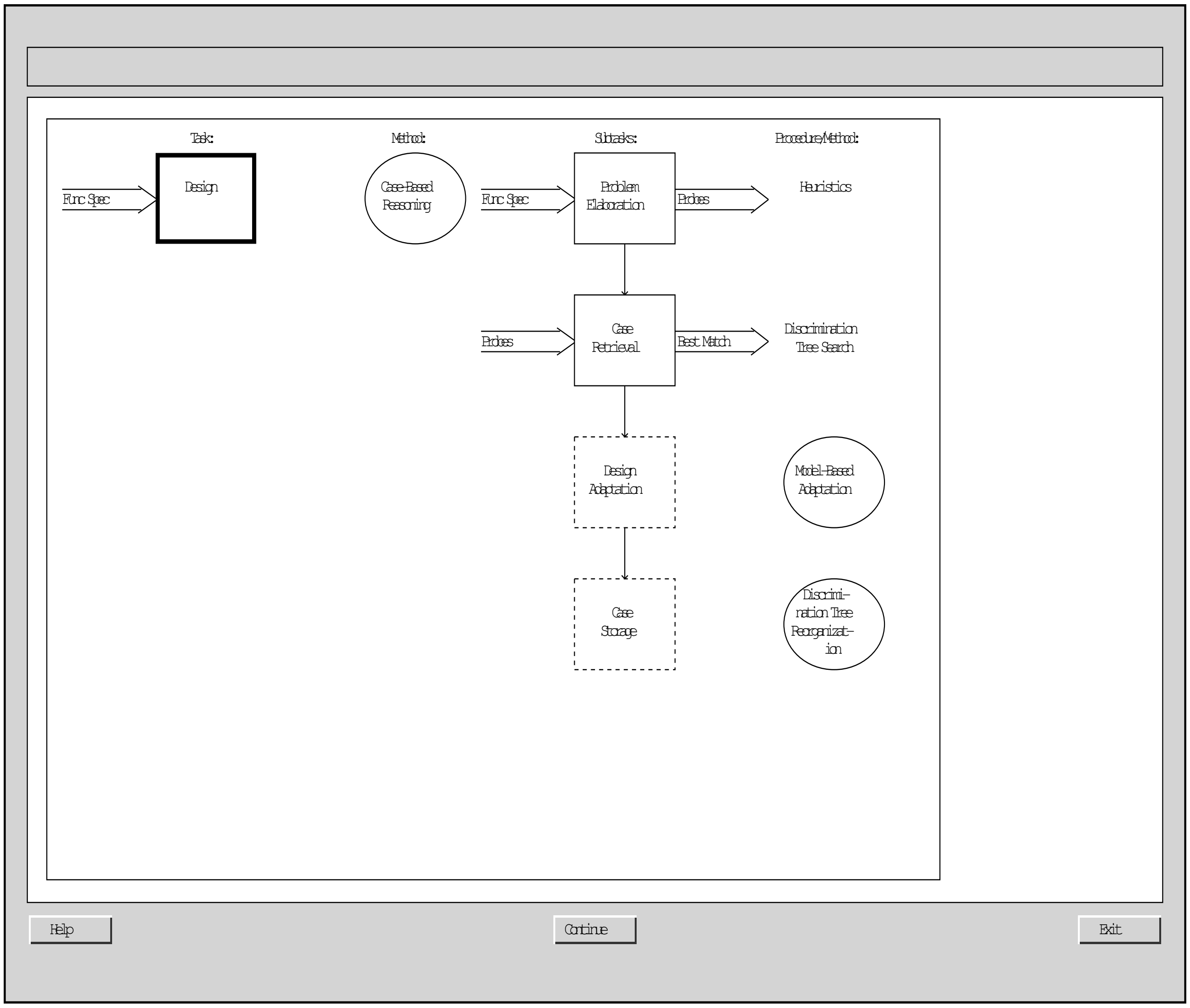


Figure B.6: Design Task Screen: After Case Retrieval

The memory of past design cases is displayed, as shown in Figure B.7. It shows the organization of the devices' models in KRITIK2's memory. The user clicks on the *Continue* button to display the best matching case retrieved from the memory.

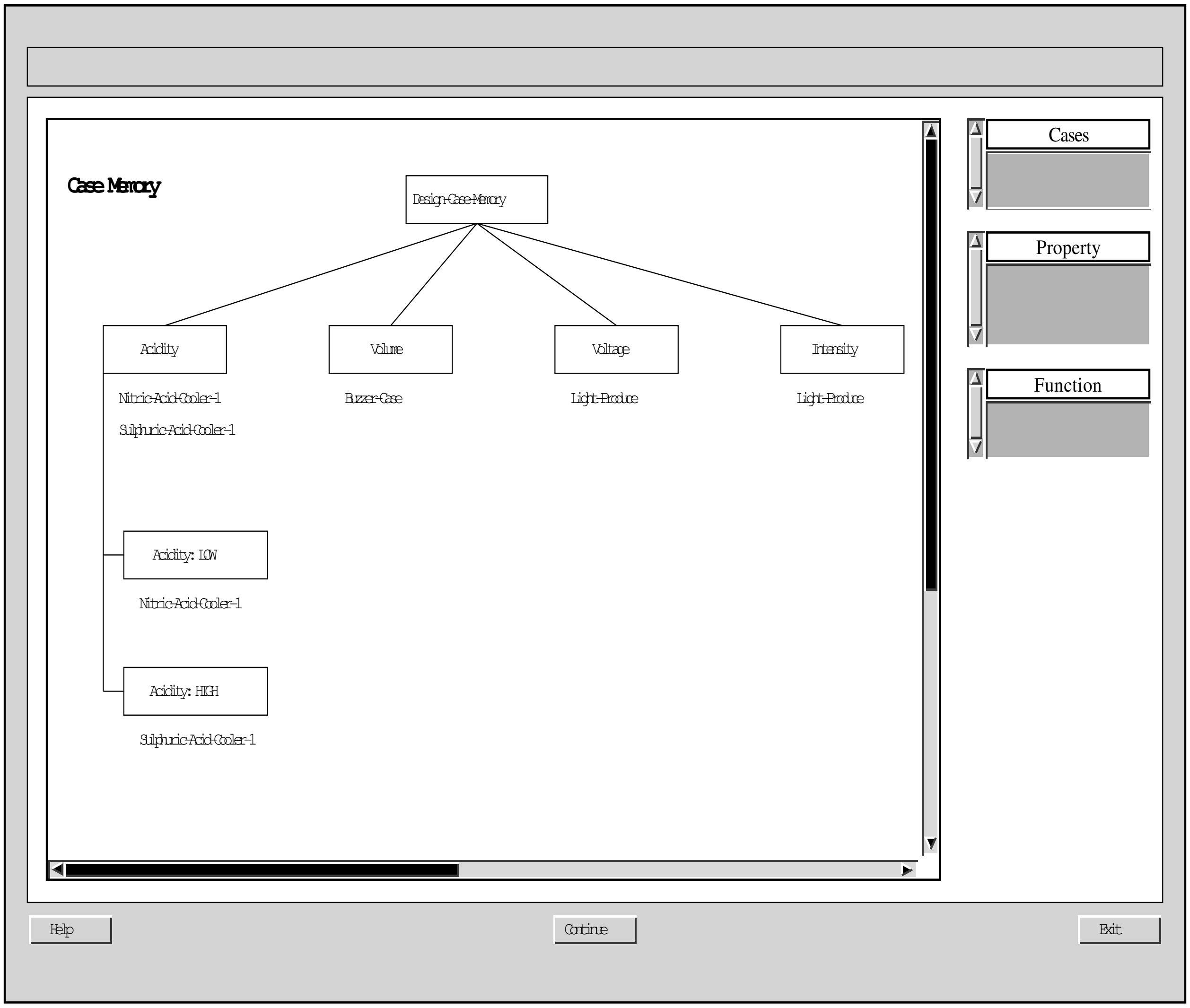


Figure B.7: KRITIK2's Memory of Design Cases

The best matching case retrieved by KRITIK2 is displayed by CANAH-CHAB. Figure B.8 shows the function of the case *Light-Produce* that was retrieved from KRITIK2's memory. The user clicks on the *Behavior* button or on *Light-Behavior* to display the internal causal behaviors of the device, as shown in Figure B.9. The user could also click on the *Continue* button to go on with the design problem solving. The result of the latter is shown in Figure B.11.

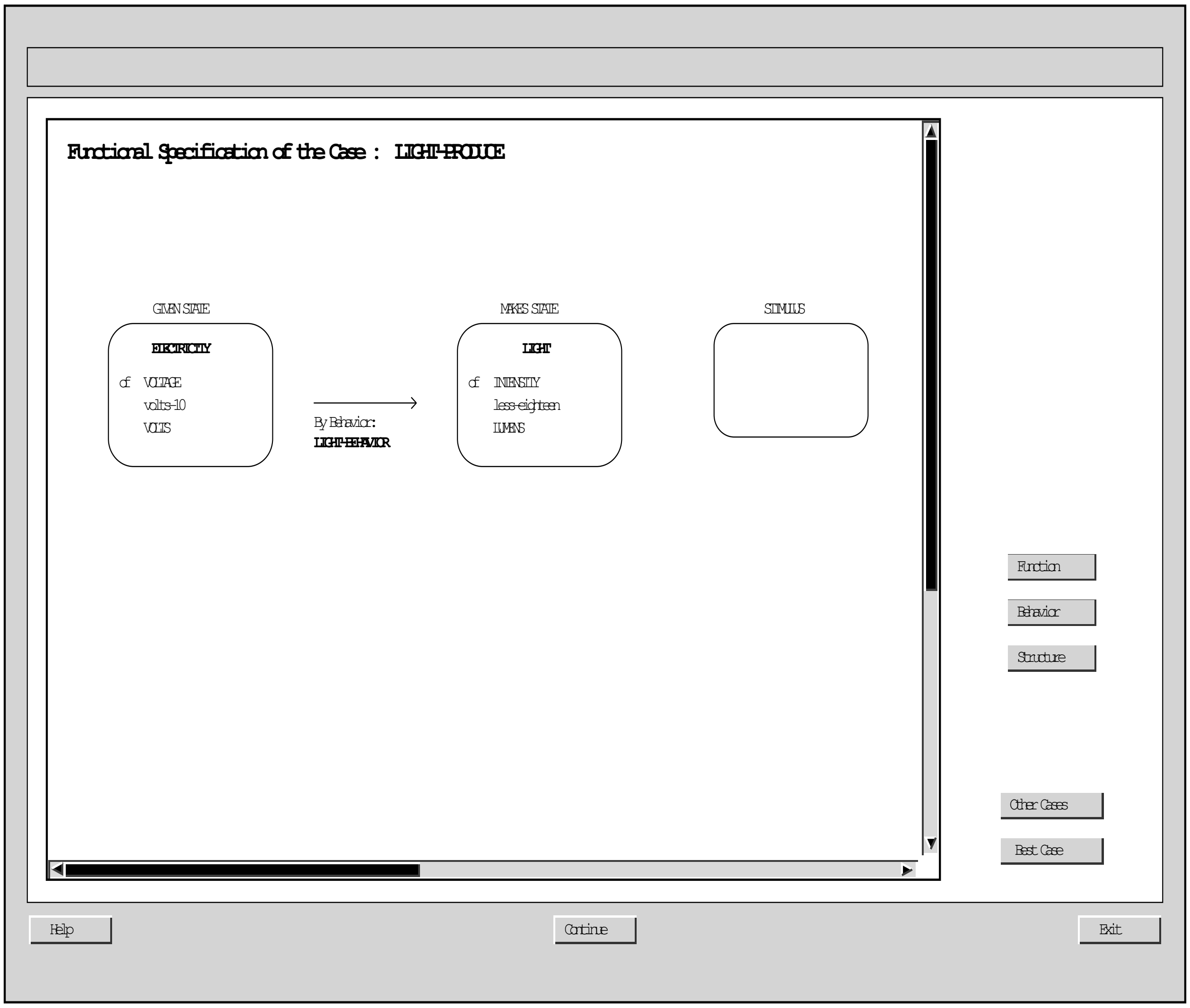


Figure B.8: Function of the Flashlight Circuit

Figure B.9 shows the behavior of the substance light in the model of the retrieved device. The user clicks on *Electricity-Behavior* to display the other behavior of the device, as shown in Figure B.10. The user could also click on the *Function* button to redisplay the function of the device shown in Figure B.8, or on the *Continue* button to go on with the design problem solving (Figure B.11).

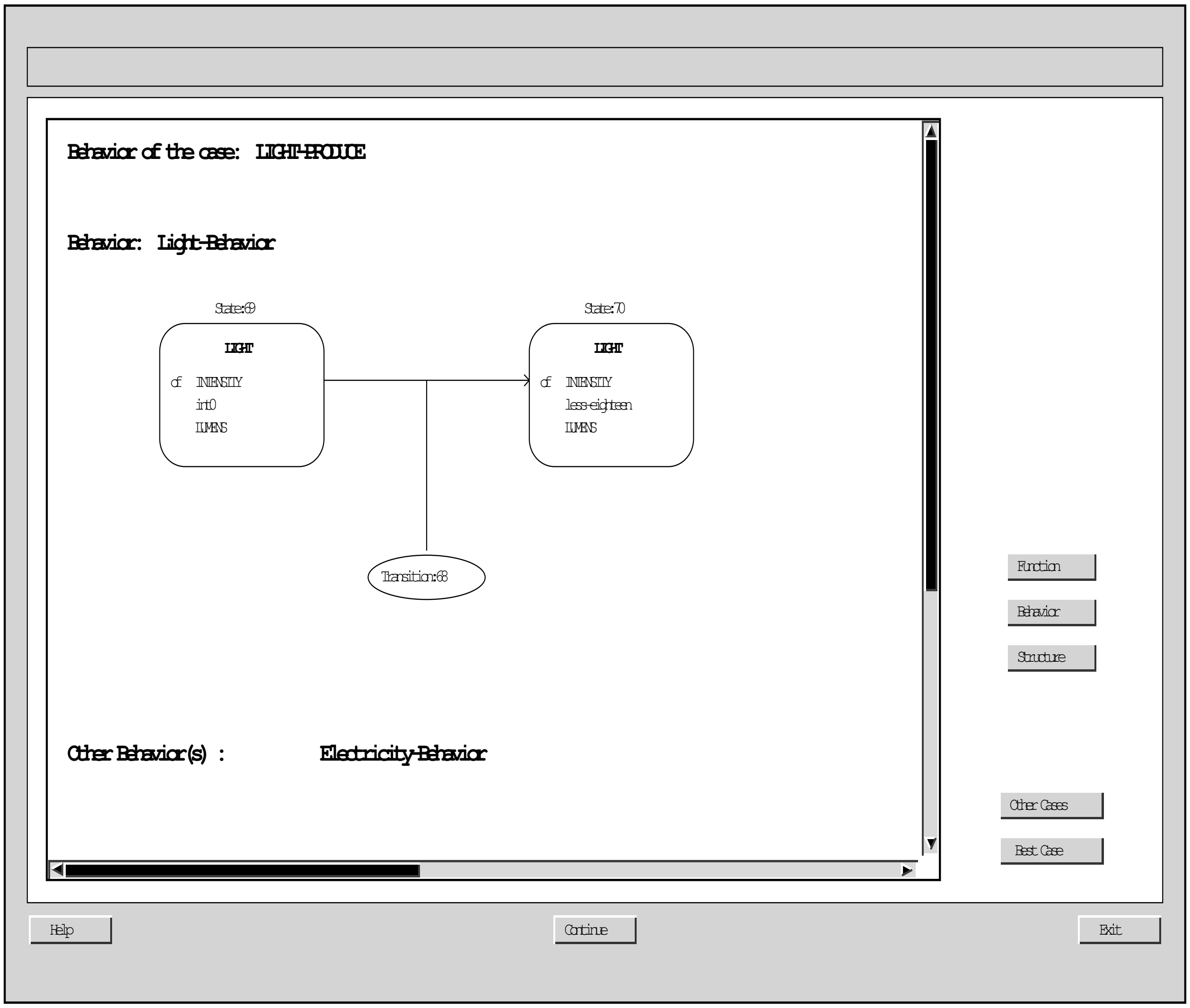


Figure B.9: Light Behavior of the Flashlight Circuit

Figure B.10 shows the behavior of the substance electricity in the model of the retrieved device. The user clicks on the *Continue* button to go on with the design problem solving (Figure B.11). The user could also click on the *Function* button to redisplay the function of the device shown in Figure B.8, or on *Light-Behavior* to re-display the other behavior of the device, as shown in Figure B.9.

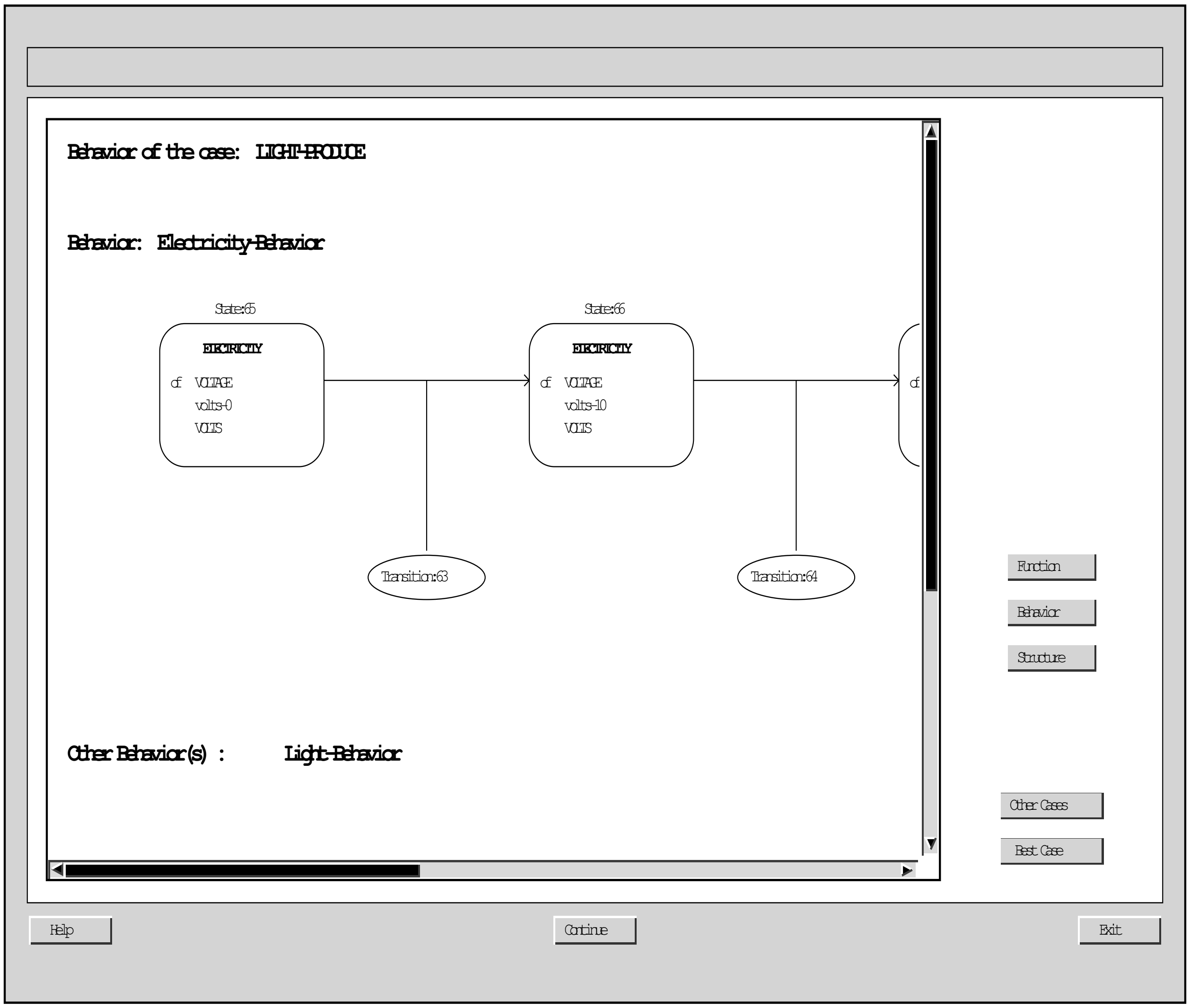


Figure B.10: Electricity Behavior of the Flashlight Circuit

Figure B.11 shows that the best matching case and the functional specification of the desired device are used as input to the *Design Adaptation* subtask. The user clicks on the *Continue* button to display how the *Design Adaptation* subtask is achieved using the *Model-Based Adaptation* method.

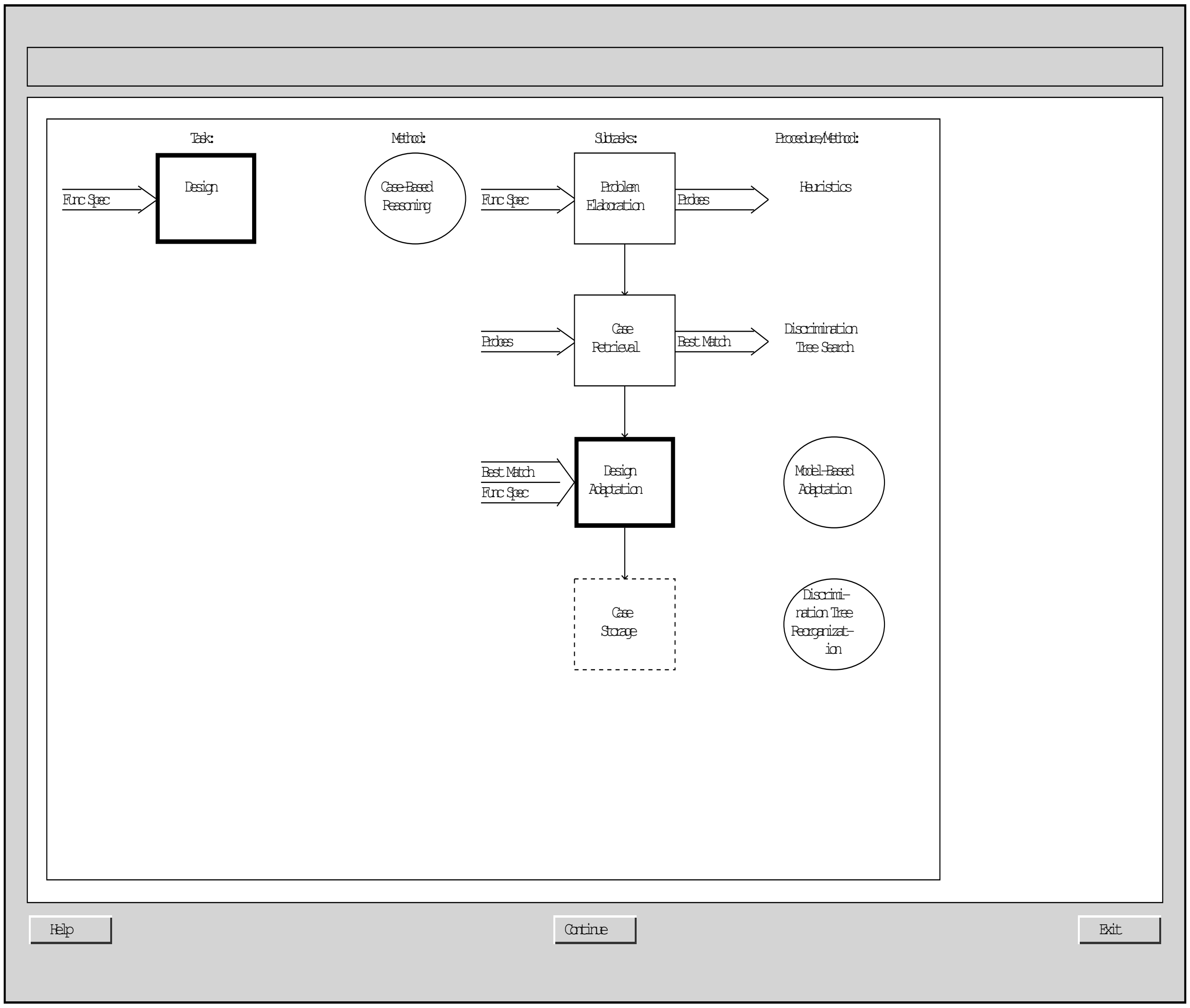


Figure B.11: Design Task Screen: Before Design Adaptation

Figure B.12 shows the task screen representing the decomposition of the *Design Adaptation* task using the *Model-Based* method. The user clicks on the *Continue* button to display the result of the *Computation of Functional Differences*.

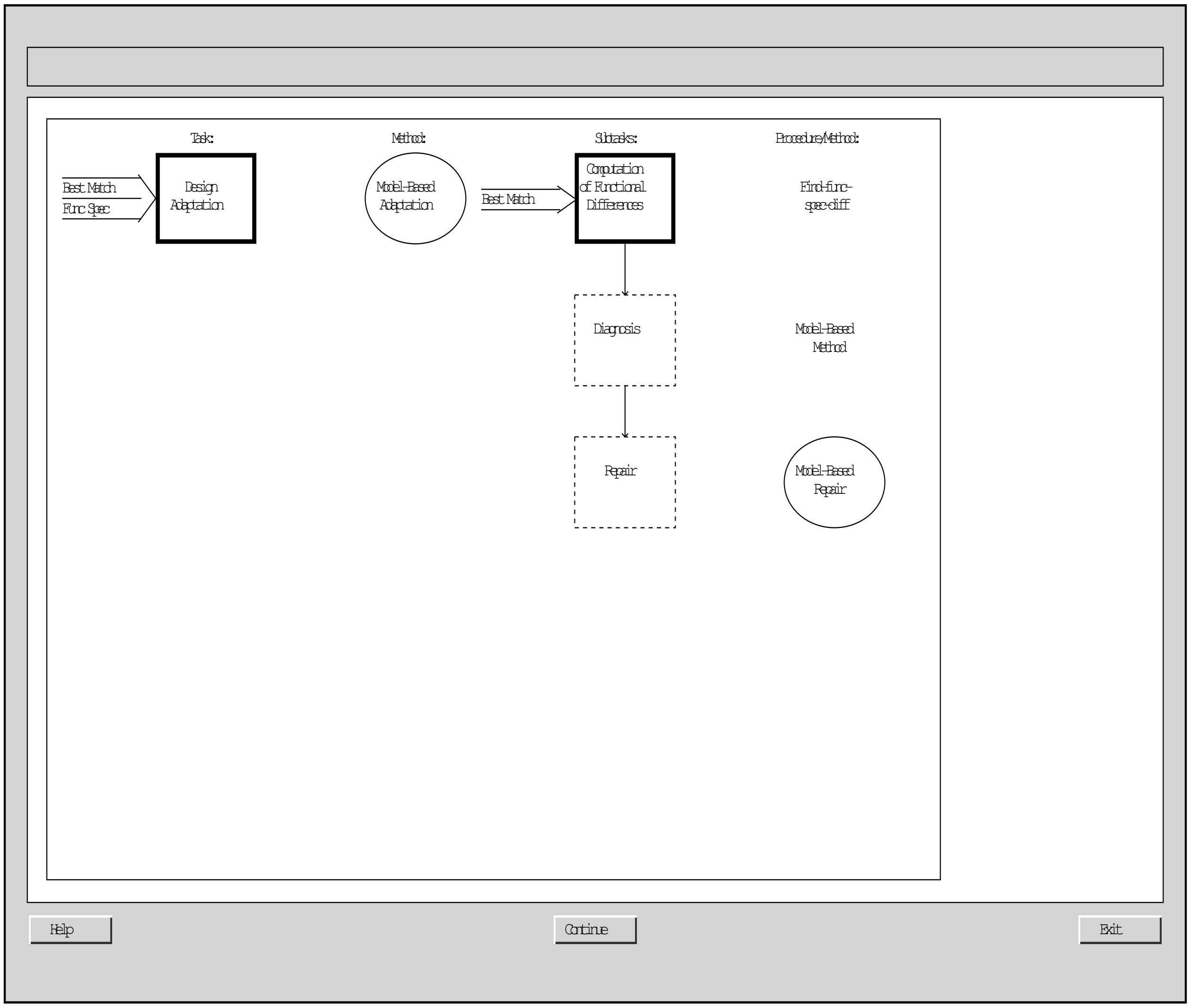


Figure B.12: Design Adaptation Task Screen: Before Computation of Differences

Figure B.13 shows that the output of the first subtask of *Design Adaptation* is a list of functional differences. The user clicks on the *Continue* button to display the functional differences.

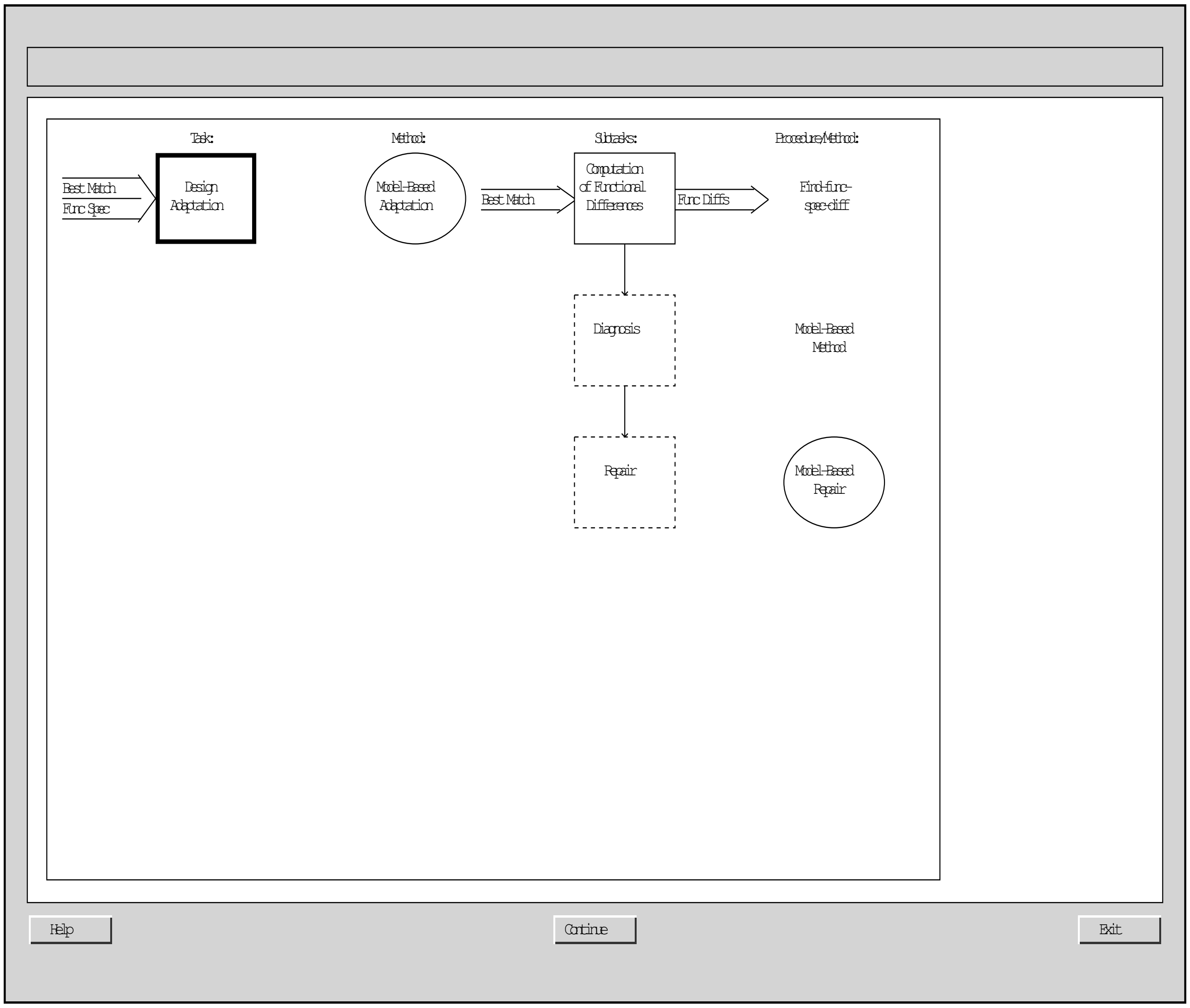
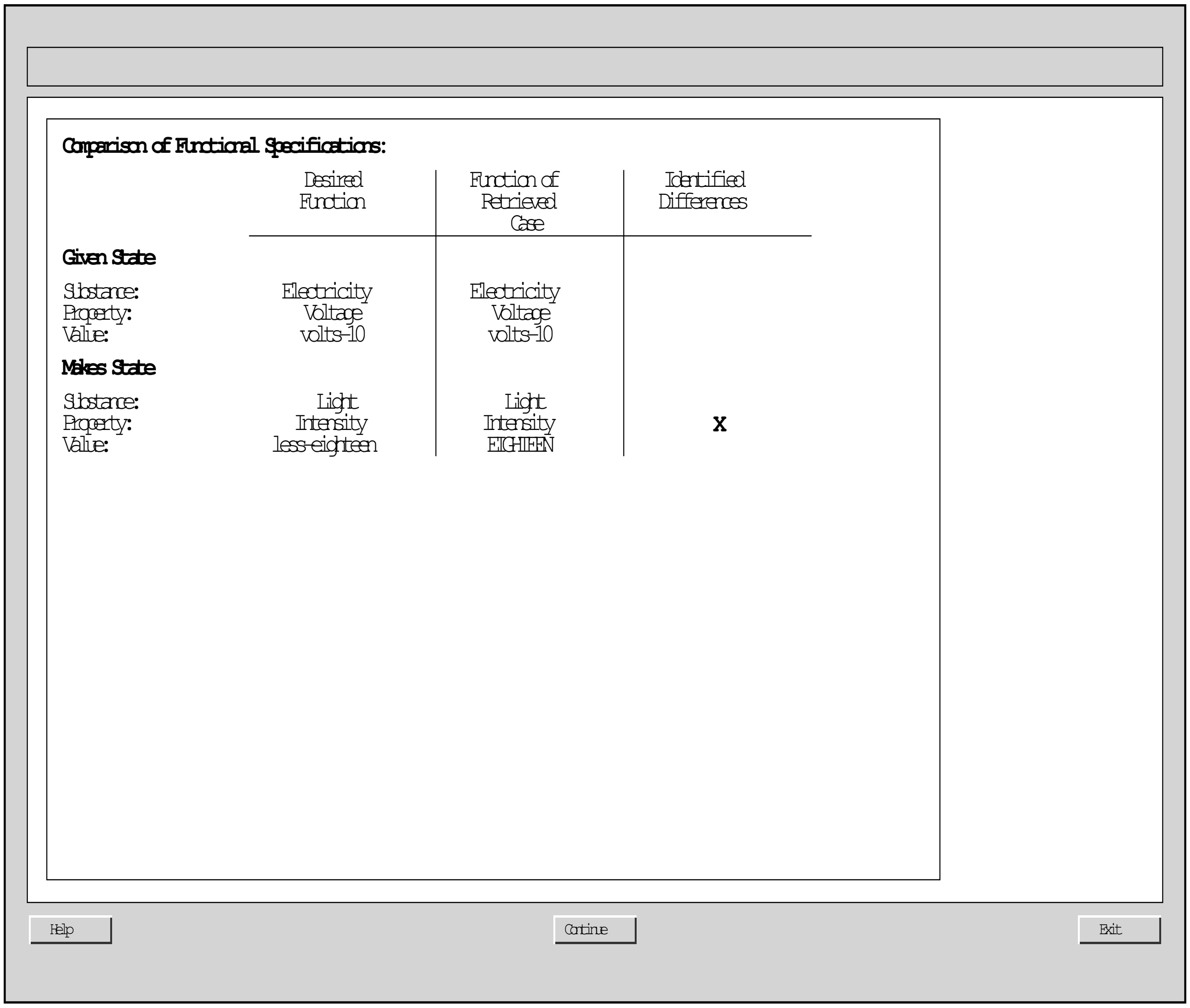


Figure B.13: Design Adaptation Task Screen: After Computation of Differences

Figure B.14 shows the result of the comparison between the desired functional specification and the function of the retrieved case. The user clicks on the *Continue* button to re-display the *Design Adaptation* screen.



Comparison of Functional Specifications:

	Desired Function	Function of Retrieved Case	Identified Differences
Given State			
Substance:	Electricity	Electricity	
Property:	Voltage	Voltage	
Value:	volts-10	volts-10	
Makes State			
Substance:	Light	Light	
Property:	Intensity	Intensity	**X**
Value:	less-eighteen	EIGHTEEN	

Figure B.14: Comparison of Functional Differences

Figure B.15 shows that the functional differences and the case retrieved are used as input to the *Diagnosis* subtask. The user clicks on the *Continue* button to display the output of the *Diagnosis* subtask.

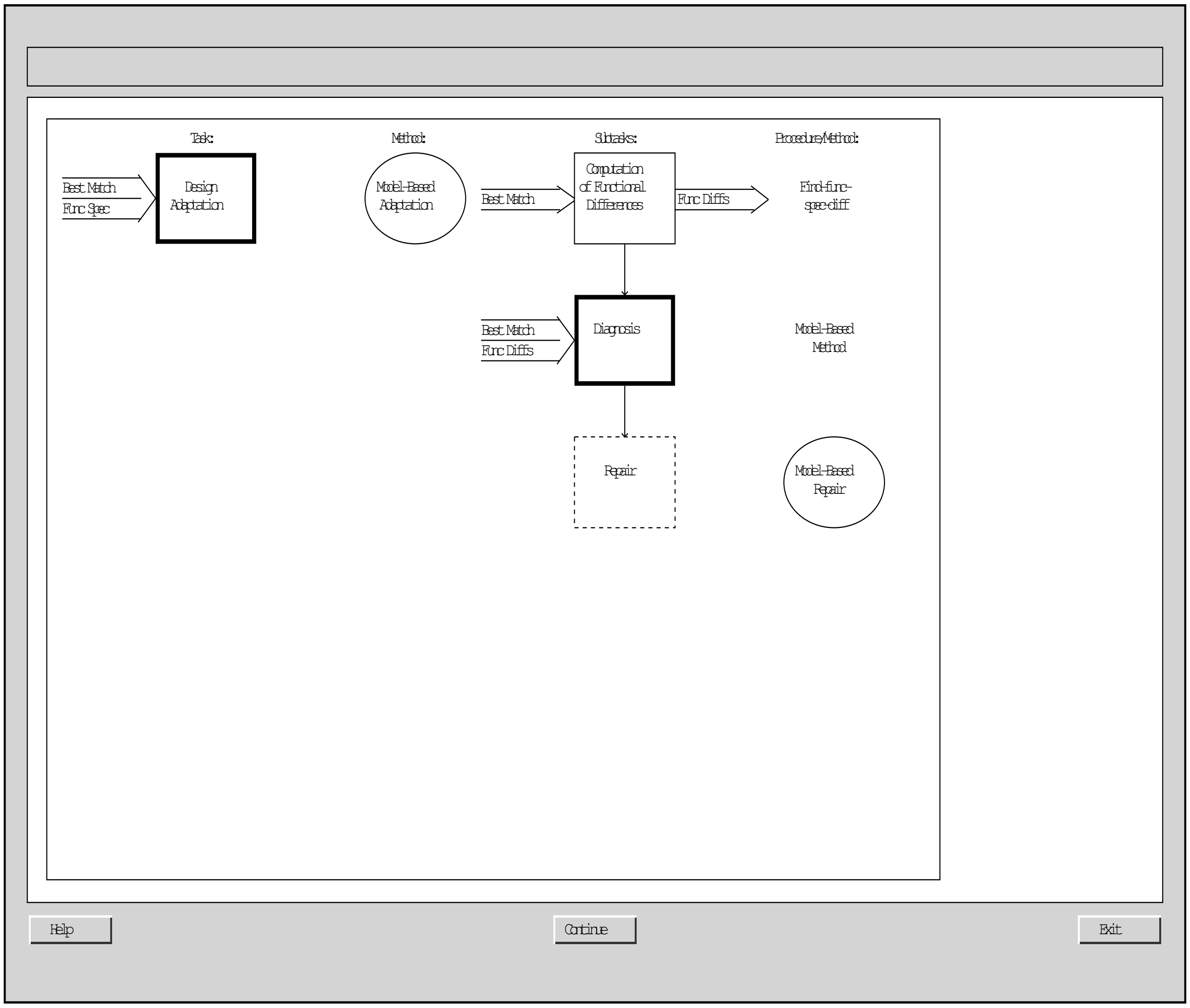


Figure B.15: Design Adaptation Task Screen: Before Diagnosis

Figure B.16 shows that the result of the *Diagnosis* subtask is a list of possible faults explaining the functional differences. The user clicks on the *Continue* button to display the output of the *Diagnosis* subtask.

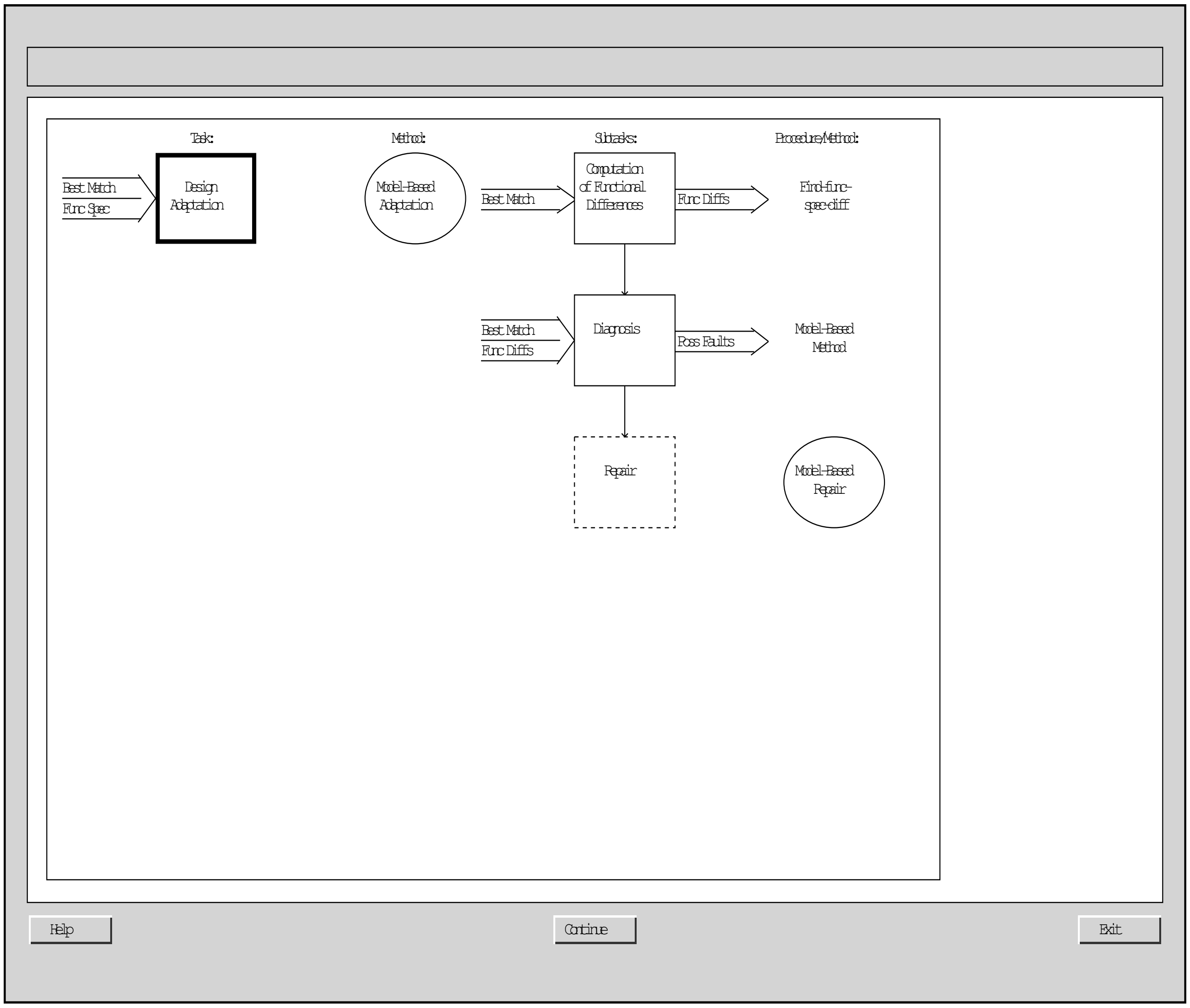
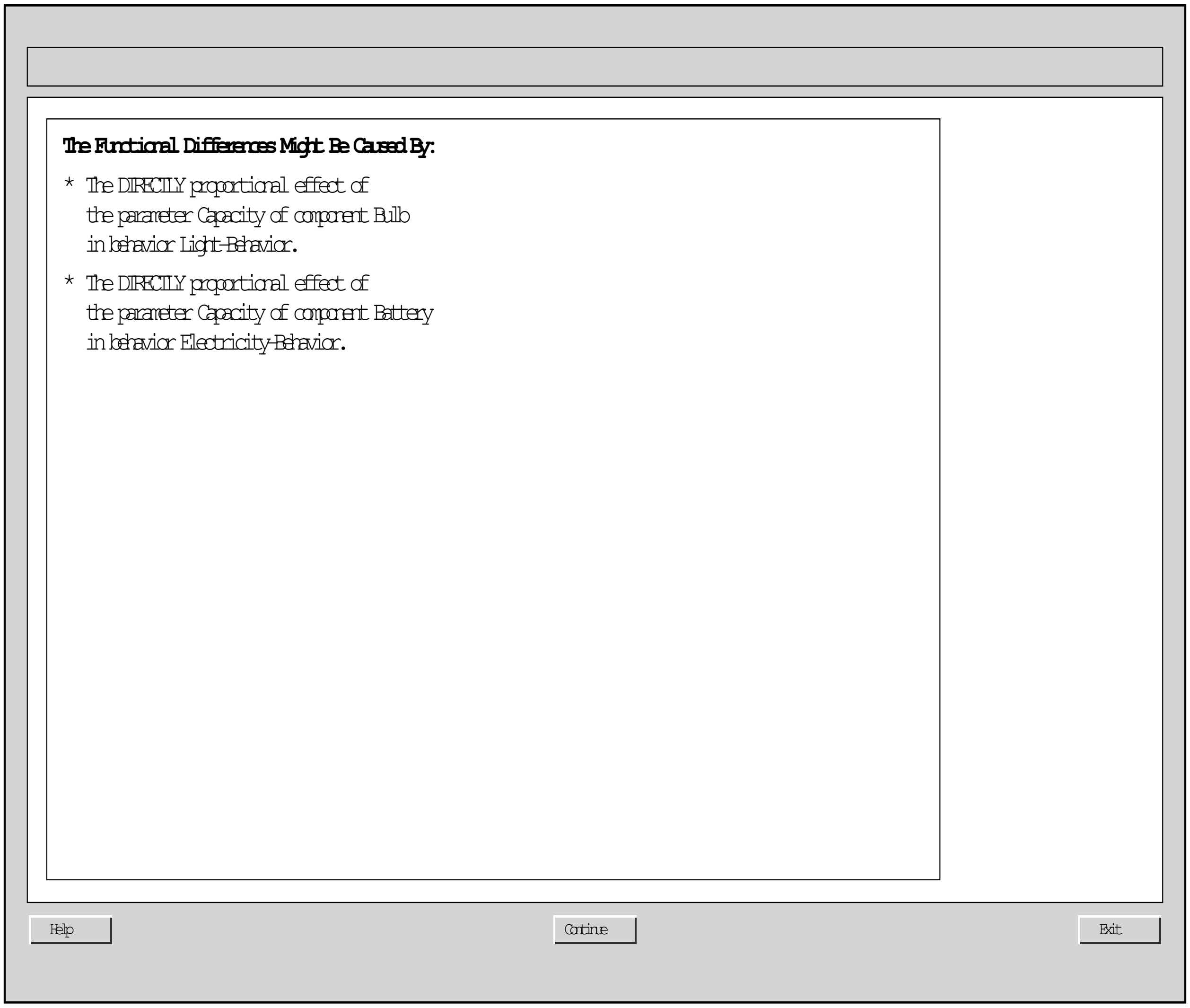


Figure B.16: Design Adaptation Task Screen: After Diagnosis

Figure B.17 shows the output of the *Diagnosis* subtask, i.e. , a list of possibles causes for the functional differences. The user clicks on the *Continue* button to display the next subtask of the *Design Adaptation* task.



The Functional Differences Might Be Caused By:

* The DIRECTLY proportional effect of
  the parameter Capacity of component Bulb
  in behavior Light-Behavior.
* The DIRECTLY proportional effect of
  the parameter Capacity of component Battery
  in behavior Electricity-Behavior.

Figure B.17: Possible Faults for the Functional Differences

Figure B.18 shows that the input to the last subtask of *Design Adaptation* consists of the functional specification, the retrieved case, the list of functional differences, and the list of possible faults. The user clicks on the *Continue* button to display how the *Repair* subtask is achieved using the *Model-Based Repair* method.

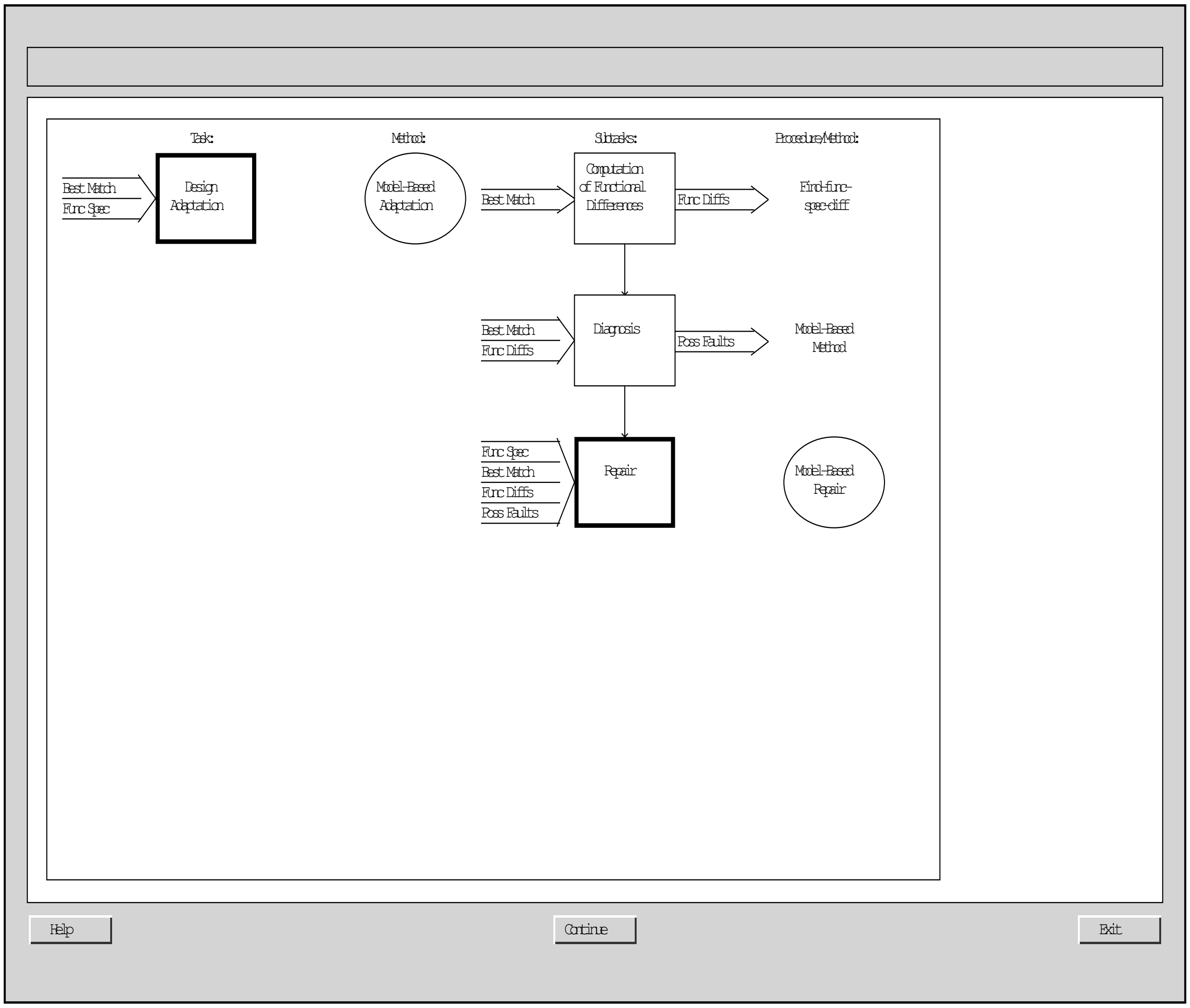


Figure B.18: Design Adaptation Task Screen: Before Repair

Figure B.19 shows how the *Repair* task is decomposed into two subtasks: *Model Revision* and *Verification*. The user clicks on the *Continue* button to display the result of the *Model revision* subtask.

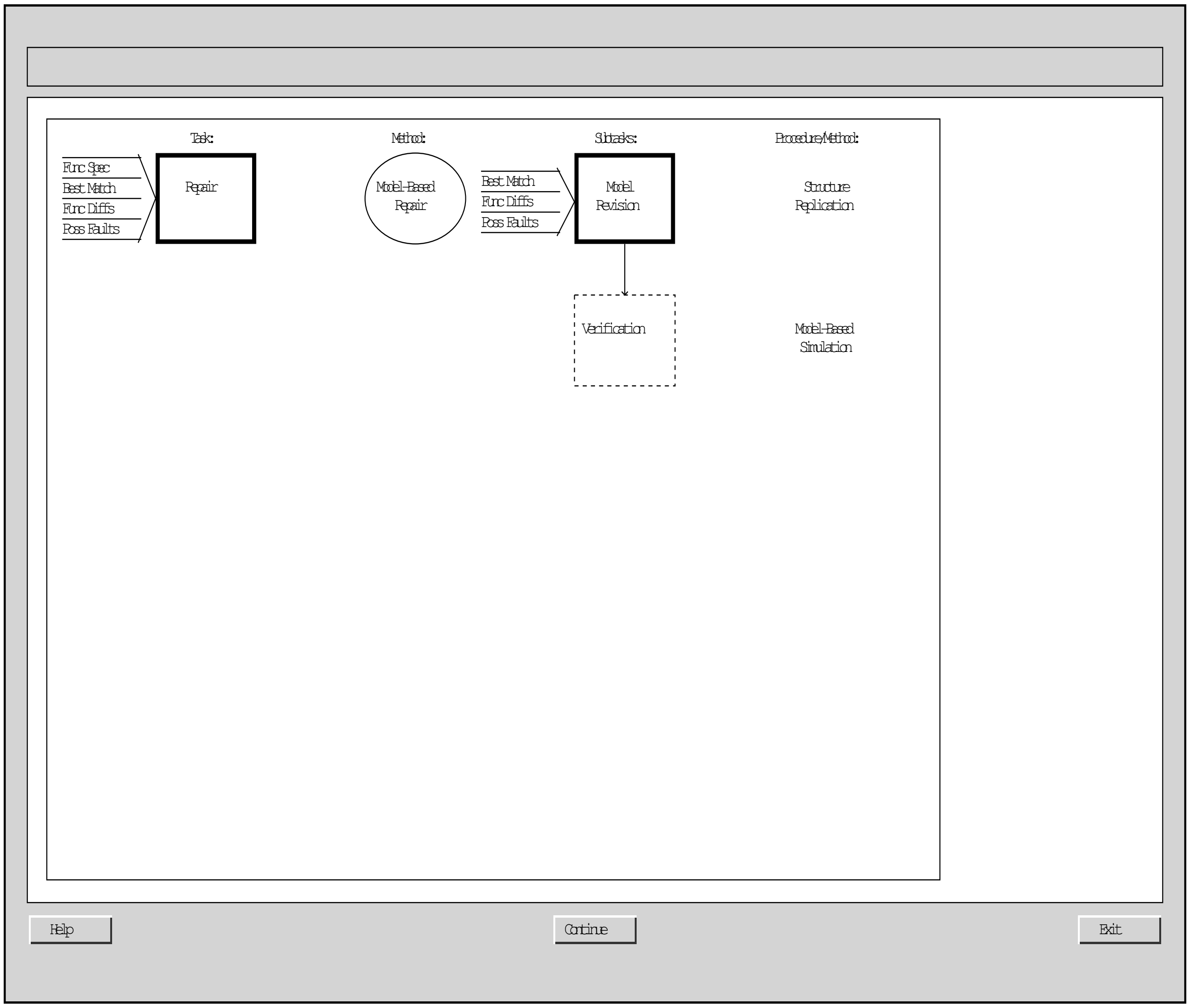


Figure B.19: Repair Task Screen: Before Model Revision

Figure B.20 shows that the output of the *Model Revision* subtask is the new SBF model describing the desired device. The user clicks on the *Continue* button to display the function of the new device.

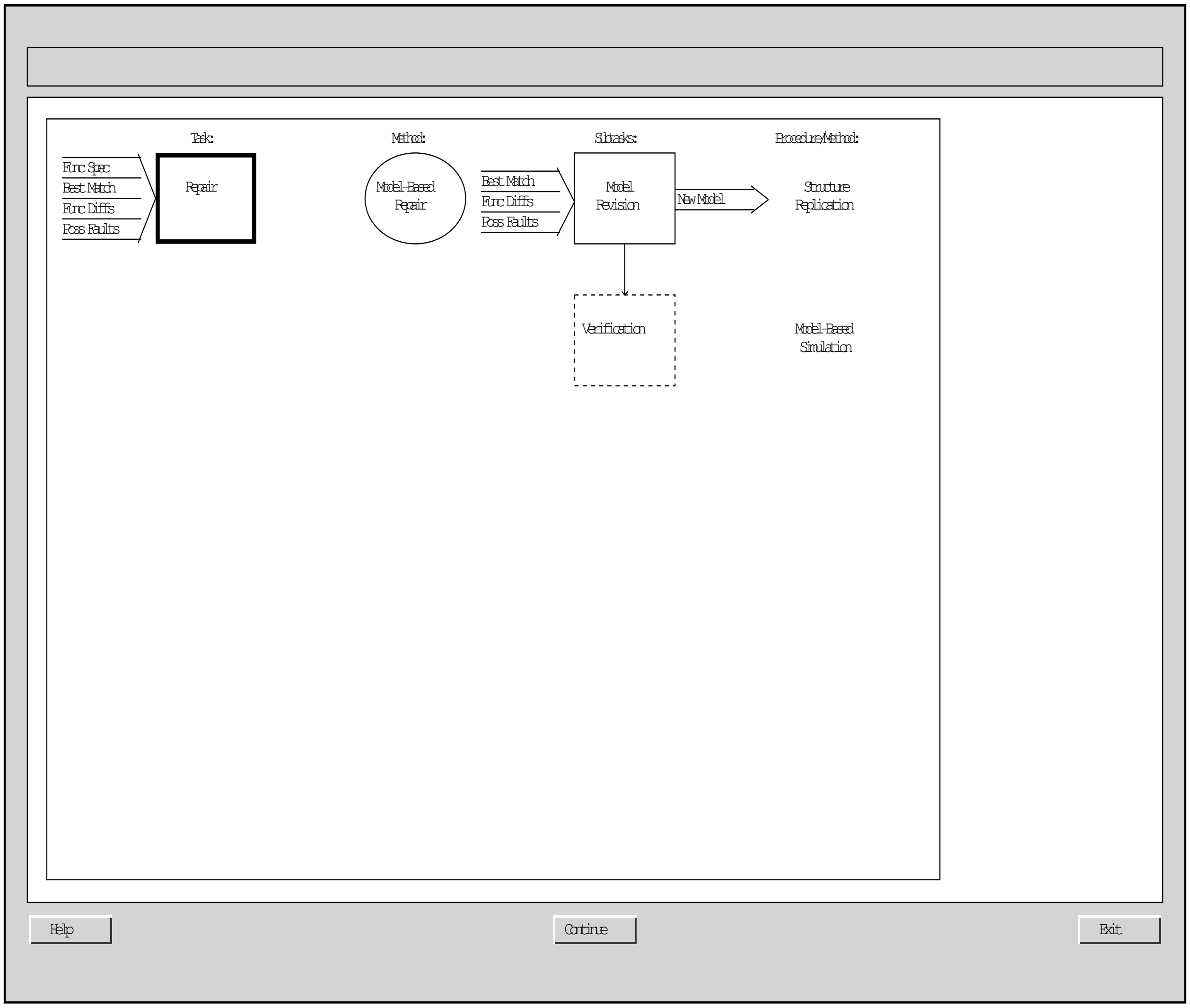
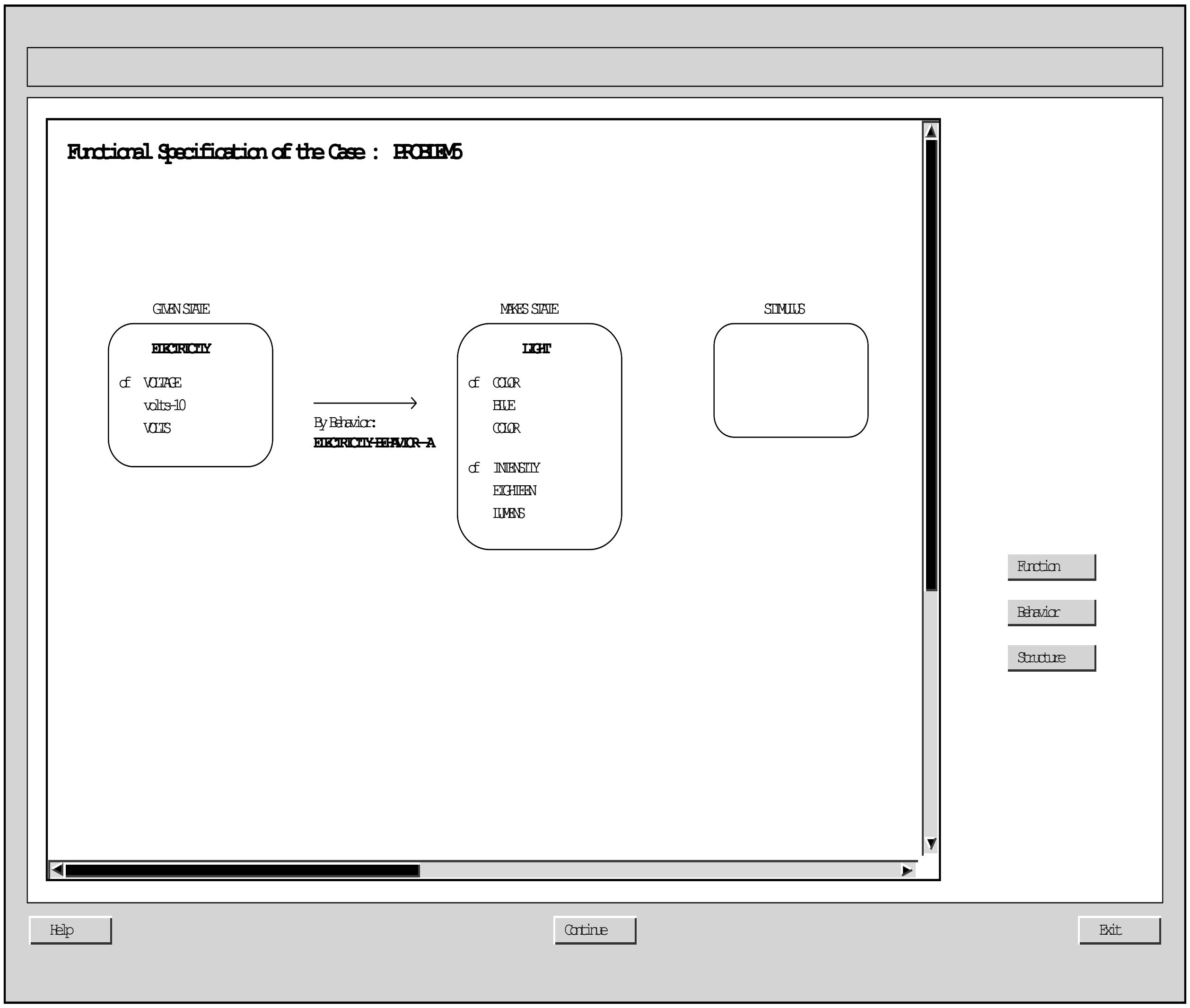


Figure B.20: Repair Task Screen: After Model Revision

Figure B.21 shows the function achieved by the new device. The user clicks on the *Behavior* button or on *Light-Behavior* to display the internal causal behaviors of the device, as it is shown in Figure B.22. The user could also click on the *Continue* button to go on with the design problem solving. The result of the latter is shown in Figure B.24.


Functional Specification of the Case : PROBLEM5

GIVEN STATE

ELECTRICITY
of VOLTAGE
volts-10
VOLTS

By Behavior:
ELECTRICITY-BEHAVIOR-A

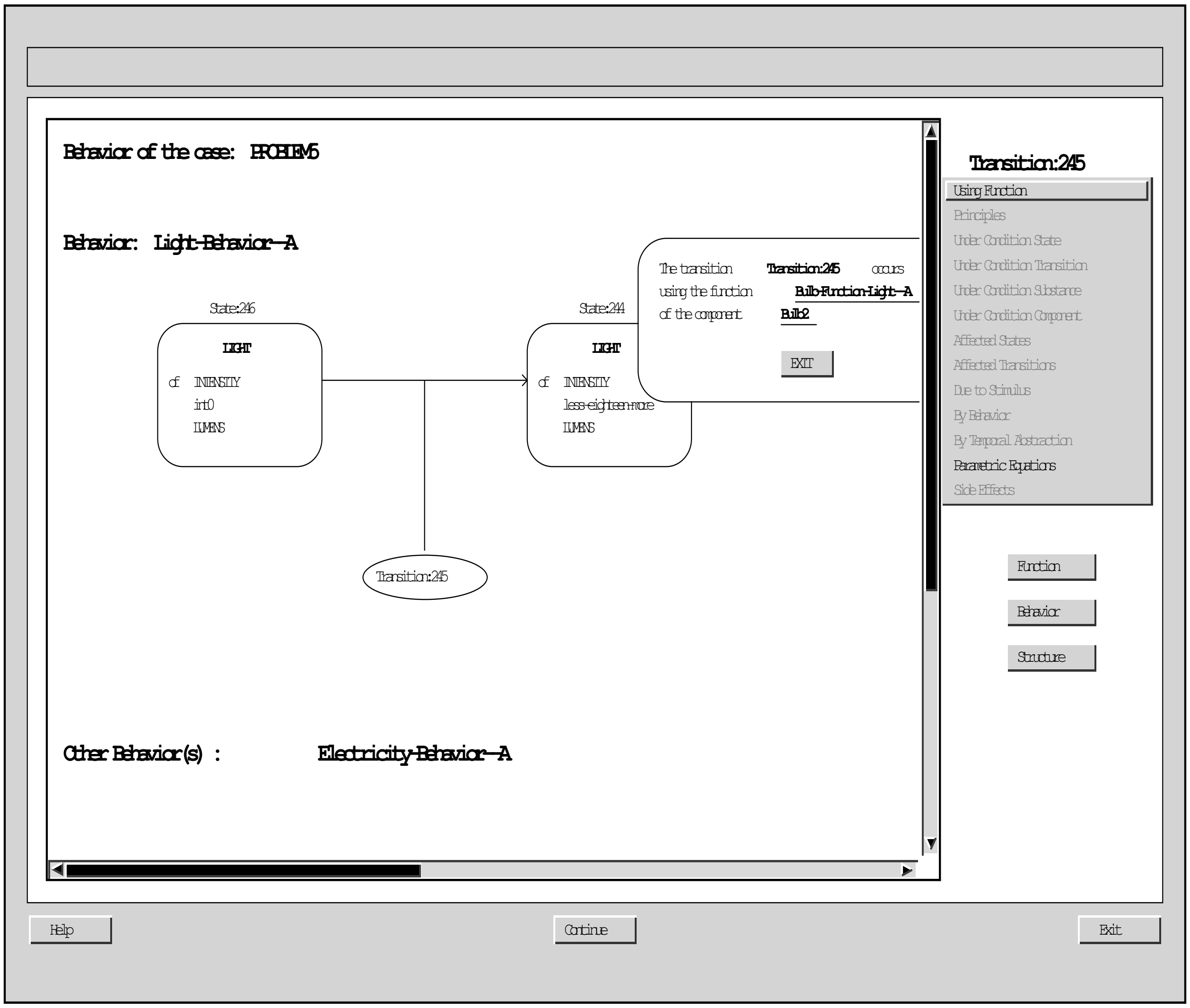
MAKES STATE

LIGHT
of COLOR
BLUE
COLOR
of INTENSITY
EIGHTEEN
LUMENS

STIMULUS

Figure B.21: Function of the New Device

Figure B.22 shows the behavior of the substance light in the model of the new device. The user clicks on *Electricity-Behavior* to display the other behavior of the device, as shown in Figure B.23. The user could also click on the *Function* button to redisplay the function of the device shown in Figure B.21, or on the *Continue* button to go on with the design problem solving (Figure B.24).


Behavior of the case: PROBLEM5

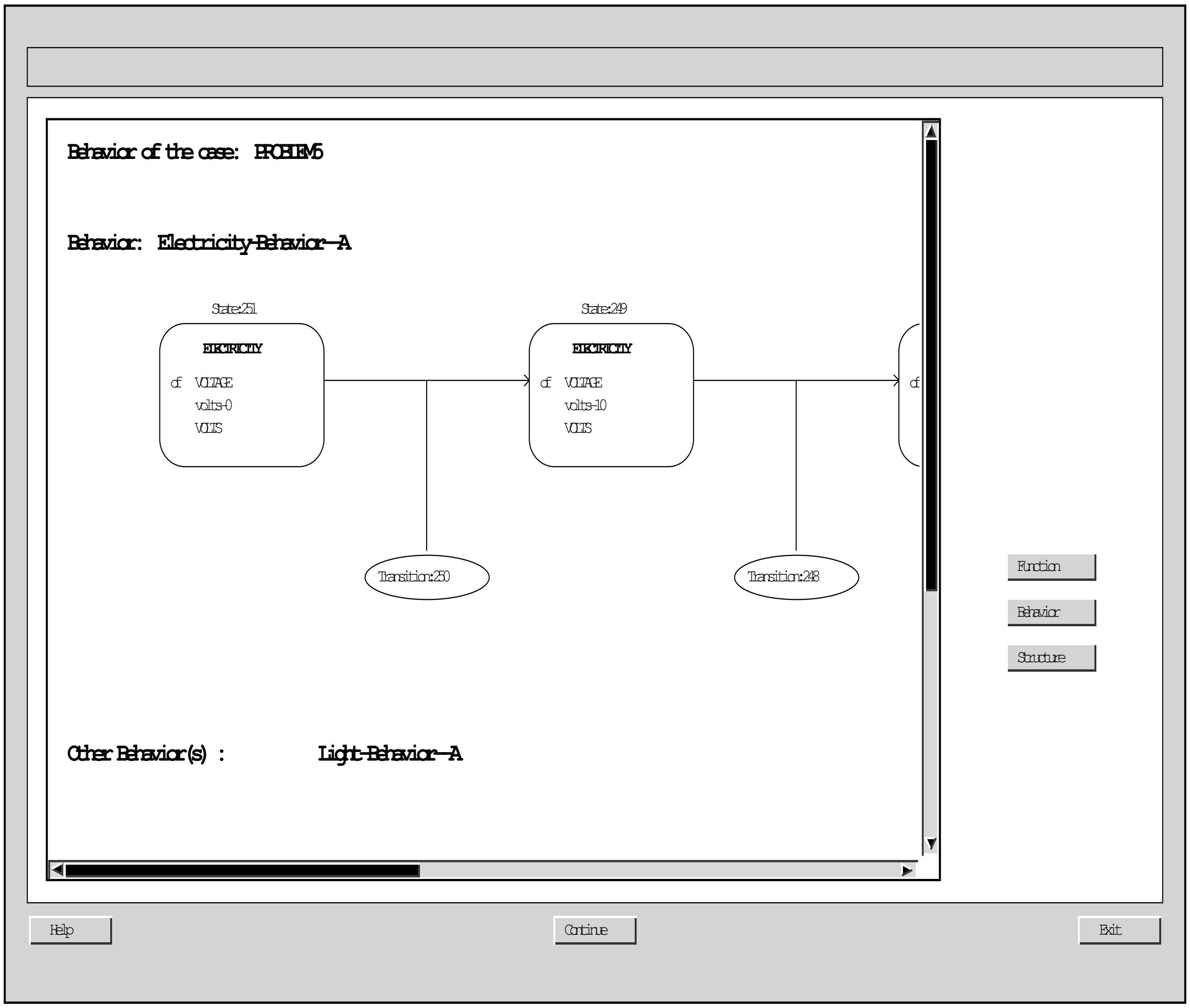
Behavior: Light-Behavior-A

The transition **Transition:245** occurs using the function **Bulb-Function-Light-A** of the component **Bulb2**

Other Behavior(s) : Electricity-Behavior-A

Figure B.22: Light Behavior of the New Device

Figure B.23 shows the behavior of the substance electricity in the model of the new device. The user clicks on the *Continue* button to go on with the design problem solving (Figure B.24). The user could also click on the *Function* button to redisplay the function of the device shown in Figure B.21, or on *Light-Behavior* to re-display the other behavior of the device, as shown in Figure B.22.


Behavior of the case: PROBLEM5

Behavior: Electricity-Behavior-A

Other Behavior(s) : Light-Behavior-A

Figure B.23: Electricity Behavior of the New Device

Figure B.24 shows that the last subtask of *Repair*, i.e., the *Verification* subtask, takes as input the desired functional specification and the new model. The user clicks on the *Continue* button to display the result of the *Verification* subtask.

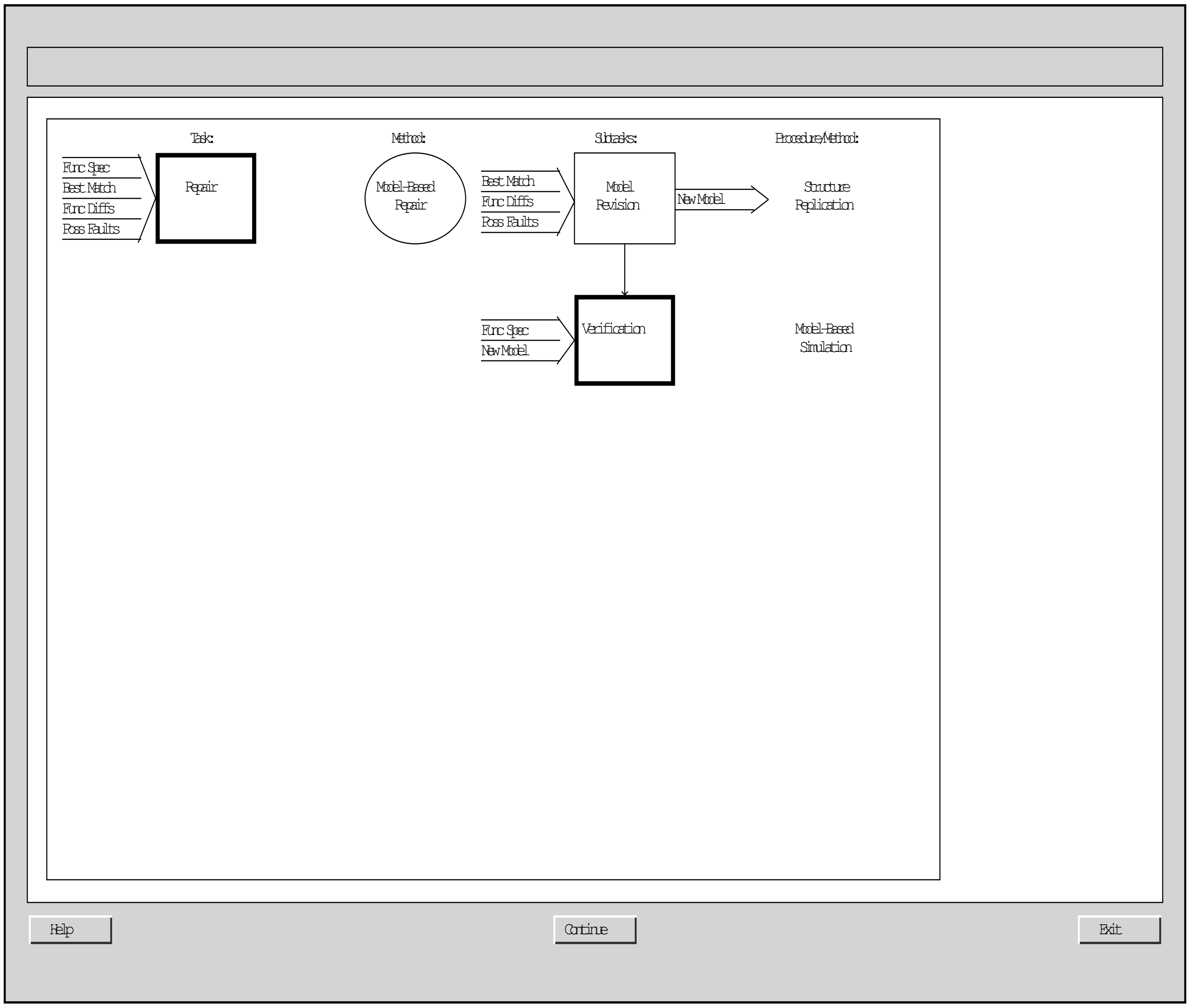


Figure B.24: Repair Task Screen: Before Verification

Figure B.25 shows that the result of the *Verification* subtask is a comparison between the desired and the achieved functions of the devices. The user clicks on the *Continue* button to display the comparison between the functions.

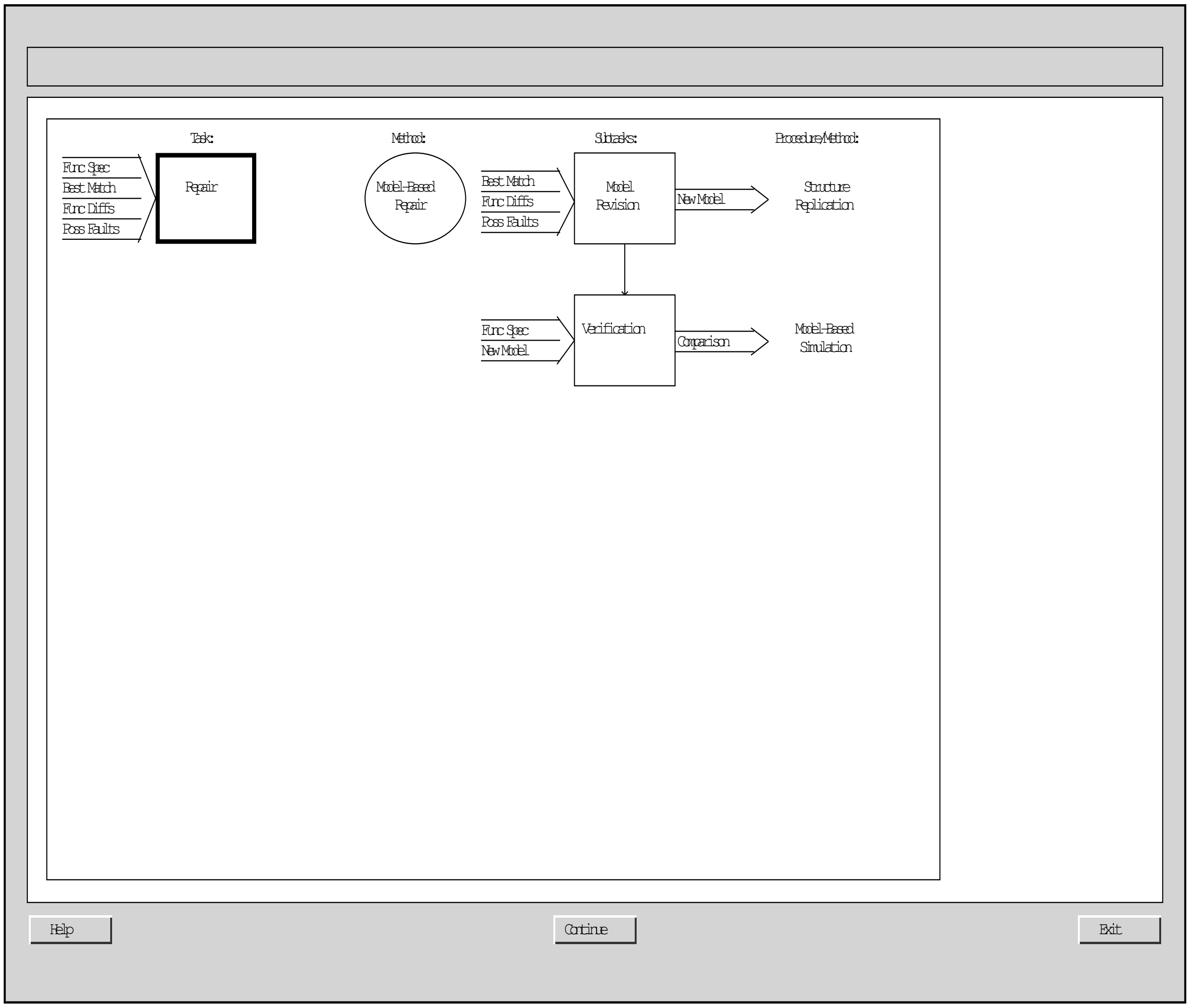
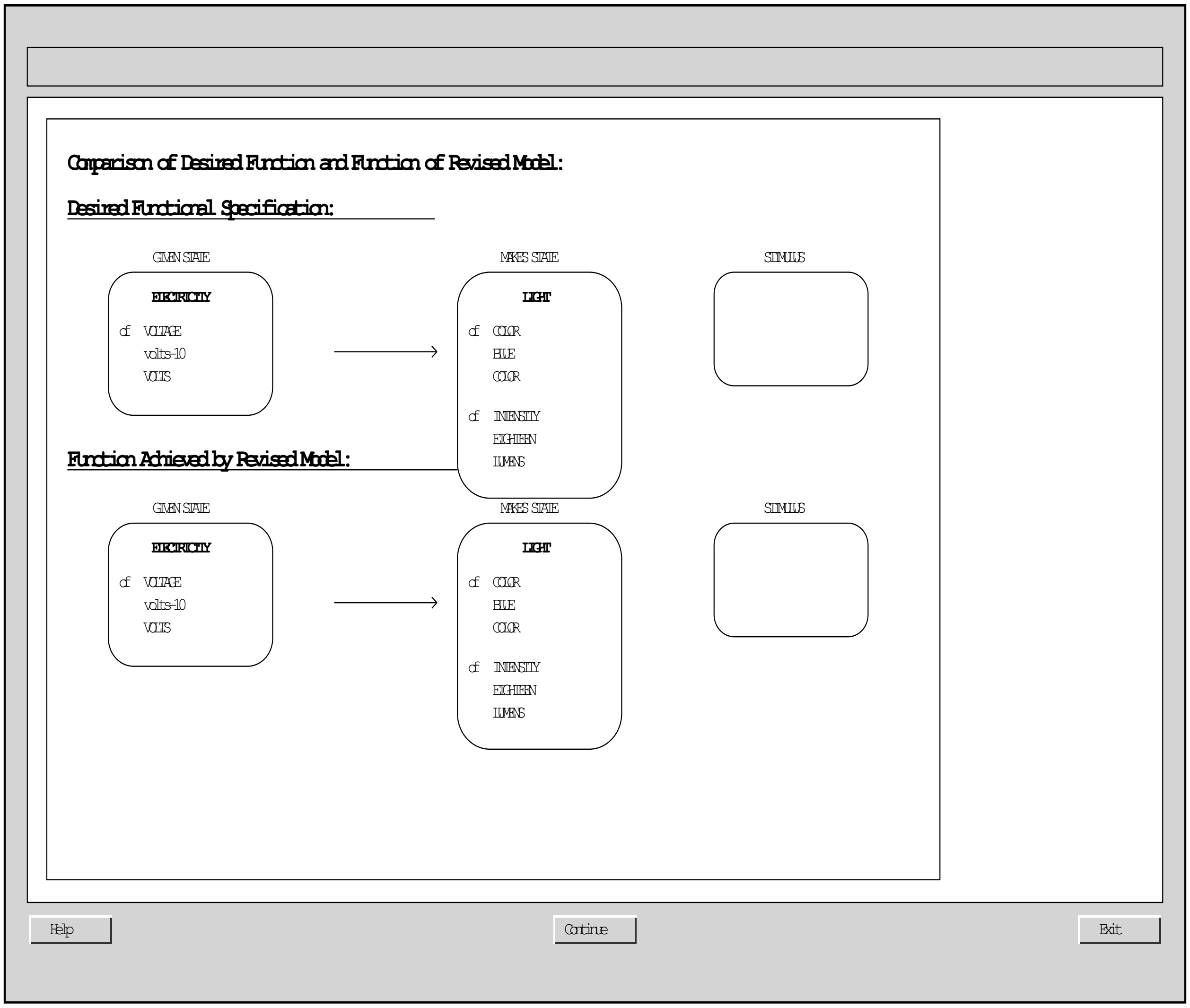


Figure B.25: Repair Task Screen: After Verification

Figure B.26 shows that the desired function of the device and the one achieved after revision of the model are equivalent. The user clicks on the *Continue* button to re-display the *Design Adaptation* task screen.



Comparison of Desired Function and Function of Revised Model:

Desired Functional Specification:

GIVEN STATE: **ELECTRICITY** of VOLTAGE volts-10 VOLTS → MAKES STATE: **LIGHT** of COLOR BLUE COLOR of INTENSITY EIGHTEEN LUMENS; STIMULUS

Function Achieved by Revised Model:

GIVEN STATE: **ELECTRICITY** of VOLTAGE volts-10 VOLTS → MAKES STATE: **LIGHT** of COLOR BLUE COLOR of INTENSITY EIGHTEEN LUMENS; STIMULUS

Figure B.26: Comparison of the Desired and Achieved Functions of the Device

Figure B.27 shows the achieved subtasks of the *Design Adaptation* task. The user clicks on the *Continue* button to display the result of the *Design Adaptation* task.

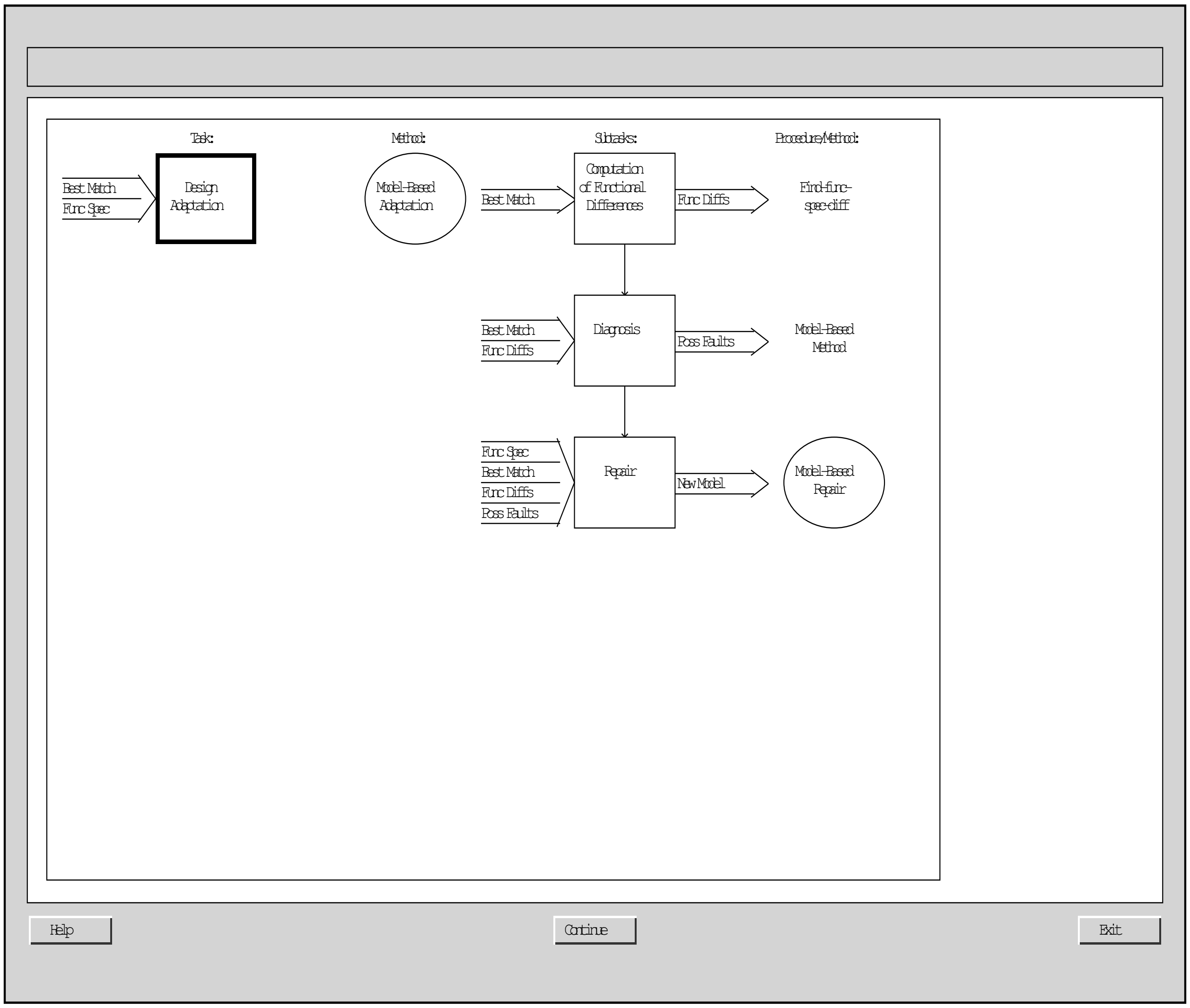


Figure B.27: Design Adaptation Task: After Repair

Figure B.28 shows that the *Design Adaptation* subtask has been completed and that its output is the new SBF model of the device. The user clicks on the *Continue* button to display the next subtask of design.

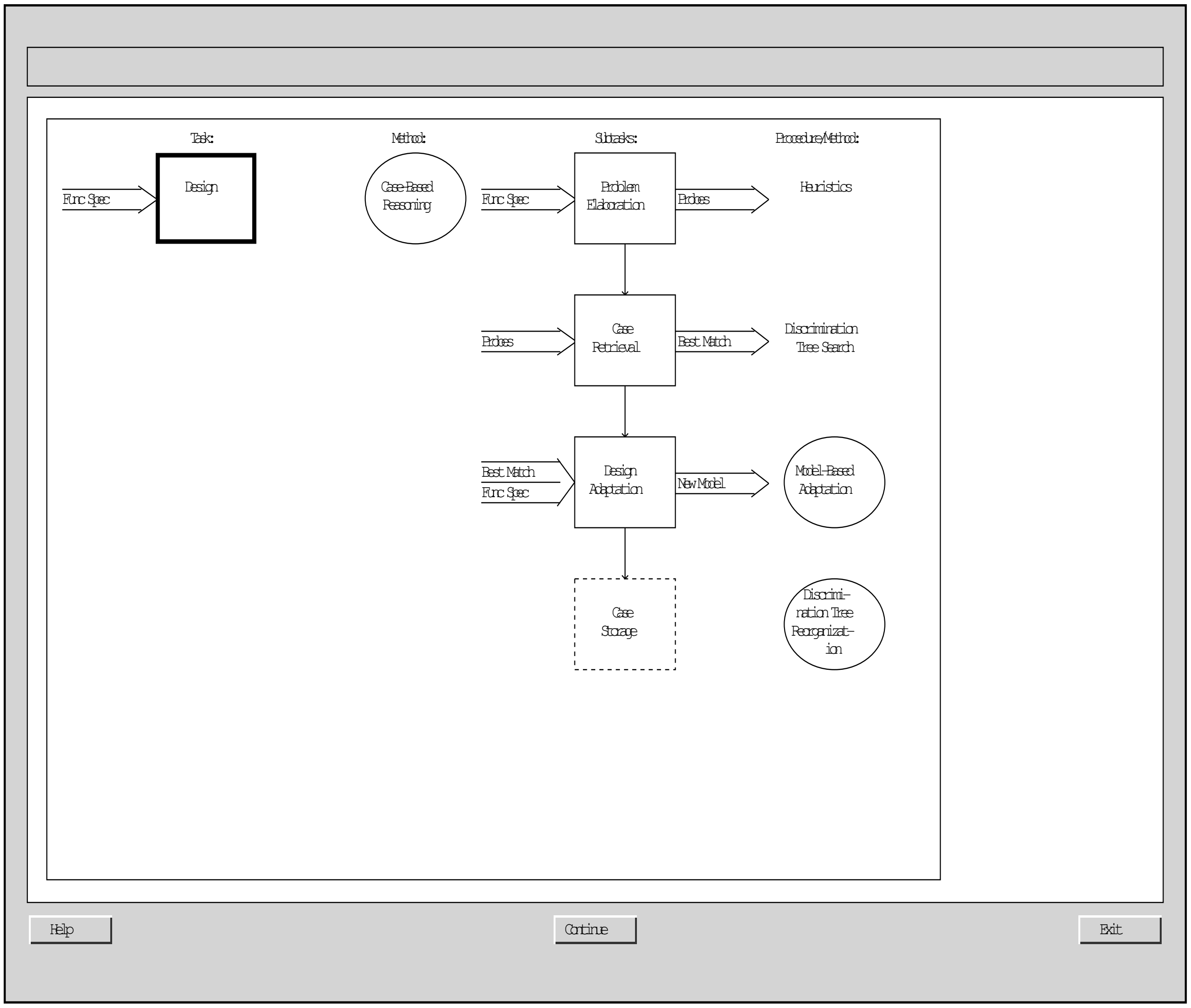


Figure B.28: Design Task Screen: After Design Adaptation

Figure B.29 shows that the new model of the device is used as input to the *Case Storage* subtask. The user clicks on the *Continue* button to display how the *case Storage* subtask is achieved using the *Discrimination Tree Reorganization* method.

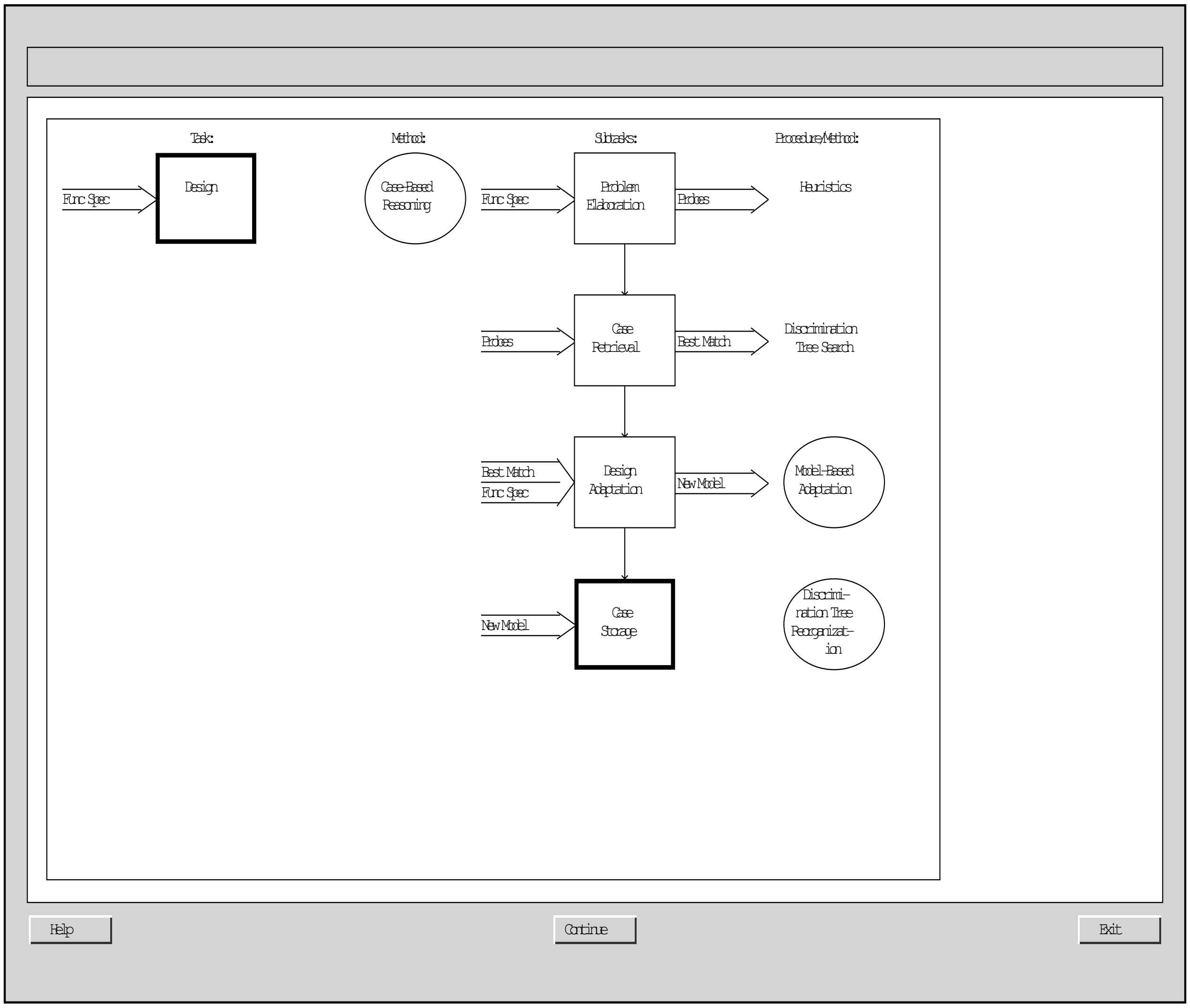


Figure B.29: Design Task Screen: Before Case Storage

Figure B.30 shows how the *case storage* task is decomposed into *Index Learning* and *Memory Replacement* subtasks. The user clicks on the *Continue* button to display the output of the *Index Learning* subtask.

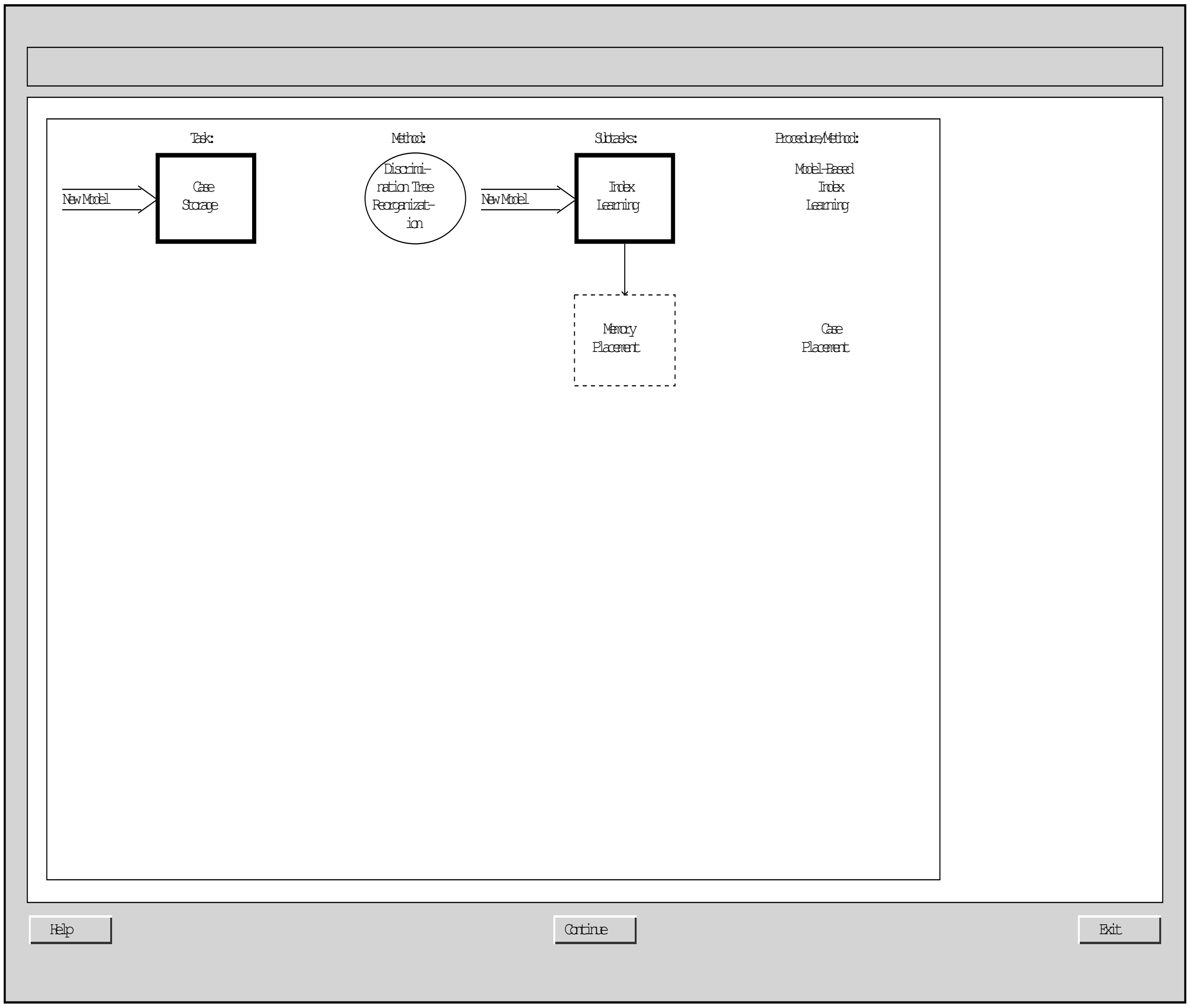


Figure B.30: Case Storage Task Screen: Before Index Learning

Figure B.31 shows that the output of the *Index Learning* subtask is a list of indices. The user clicks on the *Continue* button to display the resulting indices.

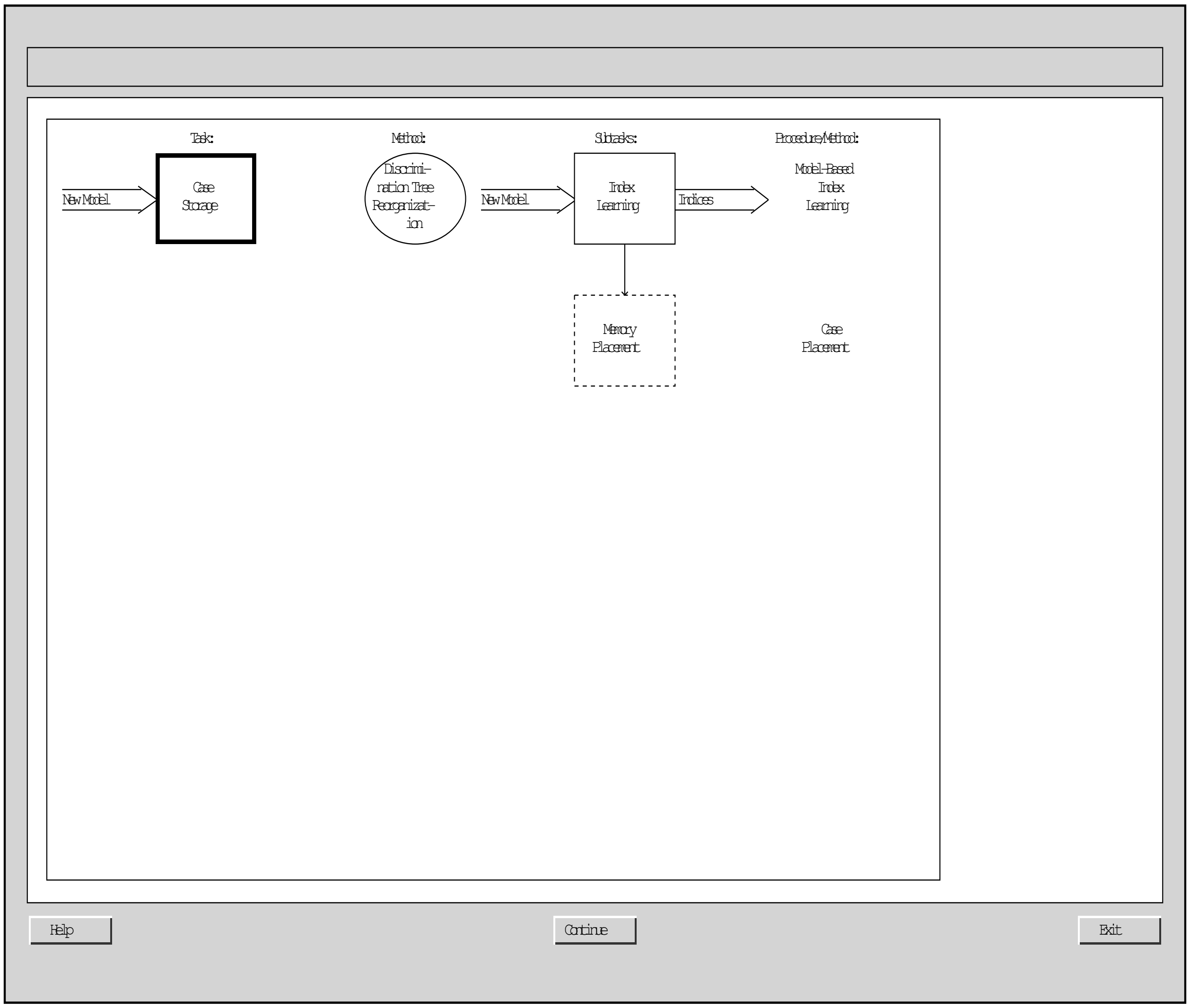
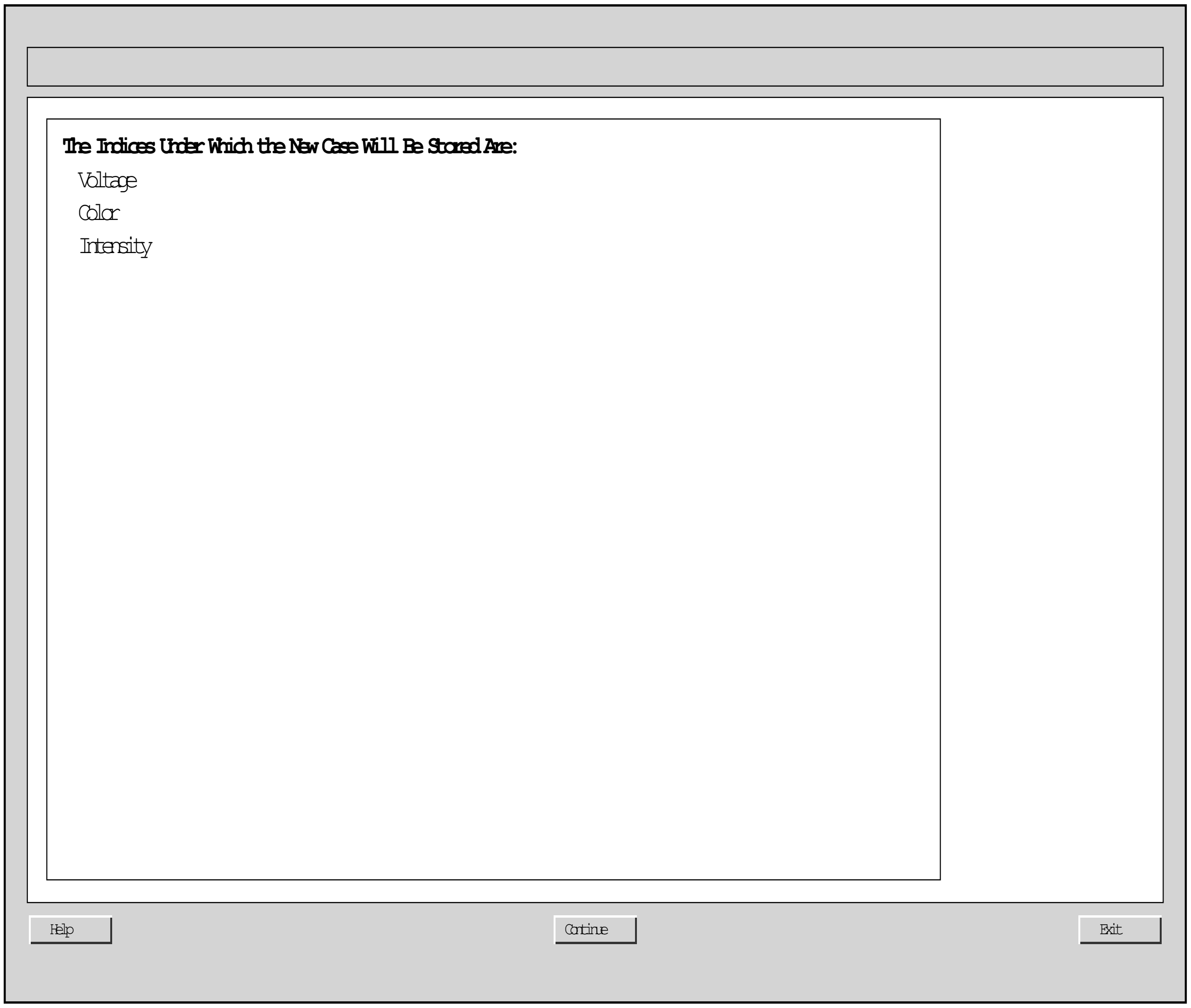


Figure B.31: Case Storage Task Screen: After Index Learning

Figure B.32 shows that the indices, under which KRITIK2 is going to store the new case in its memory of design cases, are the three properties of the substances present in the device: *Voltage*, *Color*, and *Intensity*. The user clicks on the *Continue* button to display the next subtask of the *Case Storage* task.



The Indices Under Which the New Case Will Be Stored Are:

Voltage
Color
Intensity

Figure B.32: Indices

Figure B.33 shows that the new model and the indices are used to store the new case into KRITIK2's memory of design cases. The user clicks on the *Continue* button to display the result of the *Memory Placement* subtask.

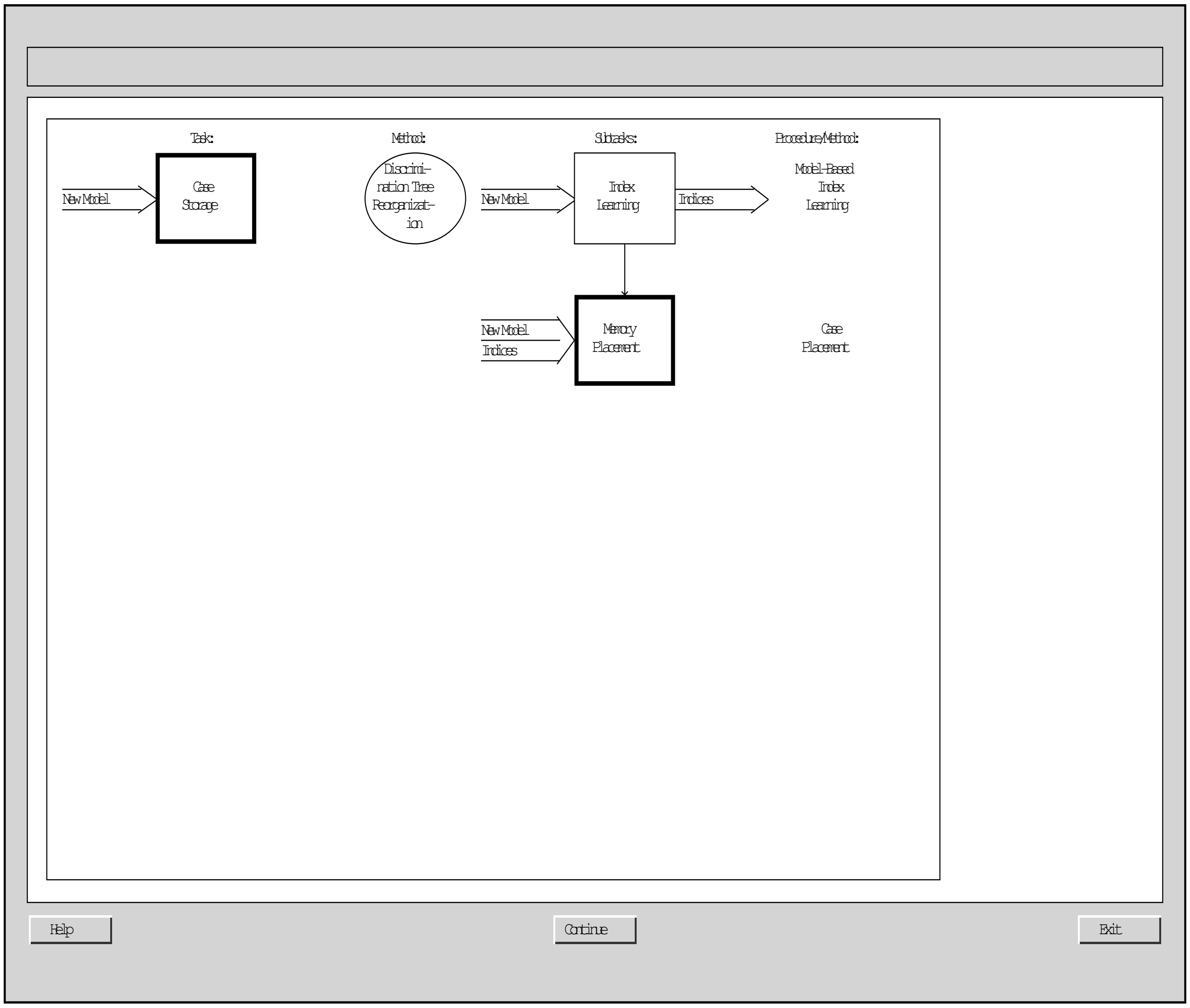


Figure B.33: Case Storage Task Screen: Before Memory Placement

Figure B.34 shows that the result of *Memory Placement* subtask is the new design case stored into KRITIK2's memory. The user clicks on the *Continue* button to display the modified memory of design cases.

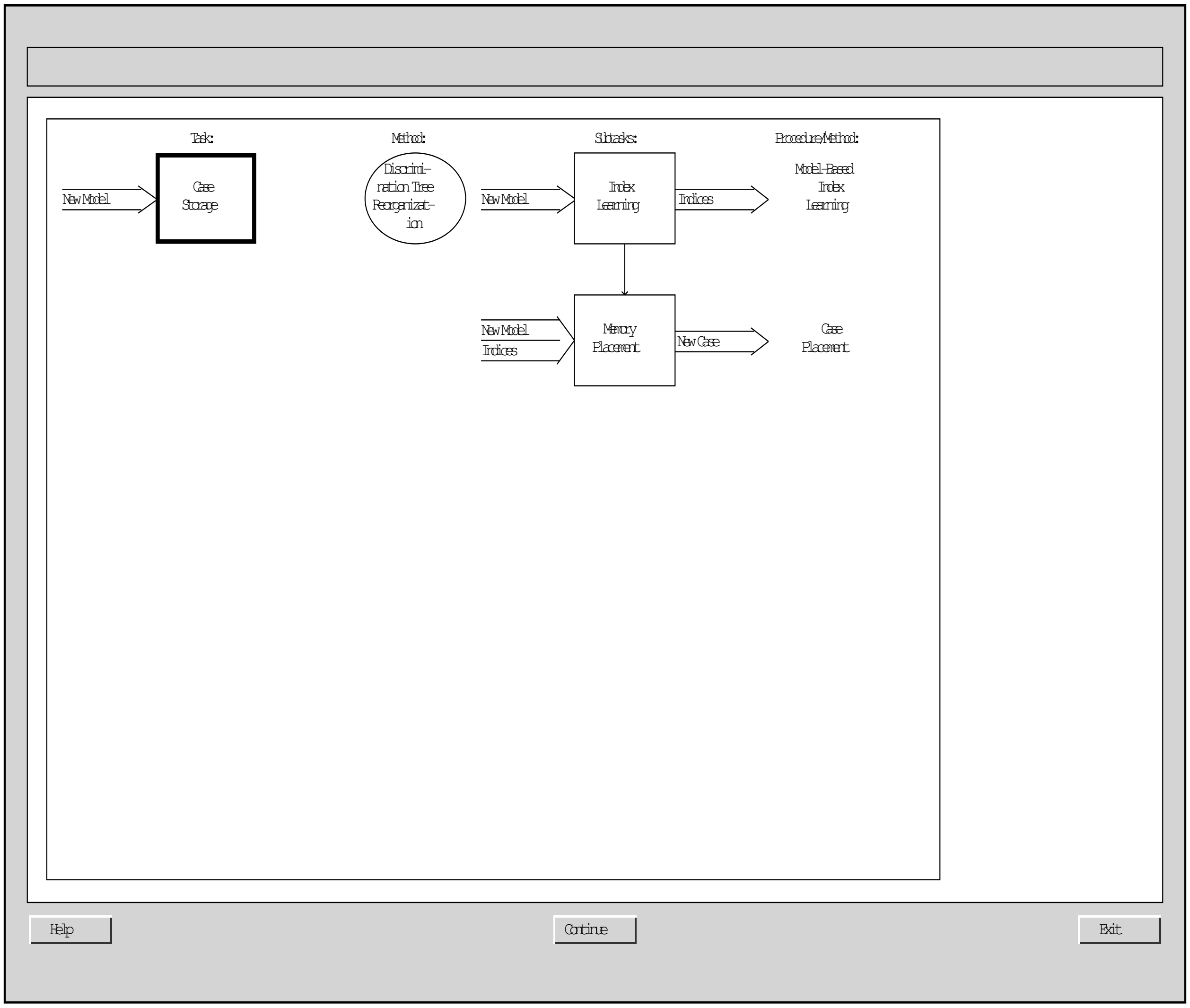


Figure B.34: Case Storage Task Screen: After Memory Placement

Figure B.35 shows the modified memory of design cases, where KRITIK2 has stored the new device (*problem5*). The user clicks on the *Continue* button to display the complete design task screen.

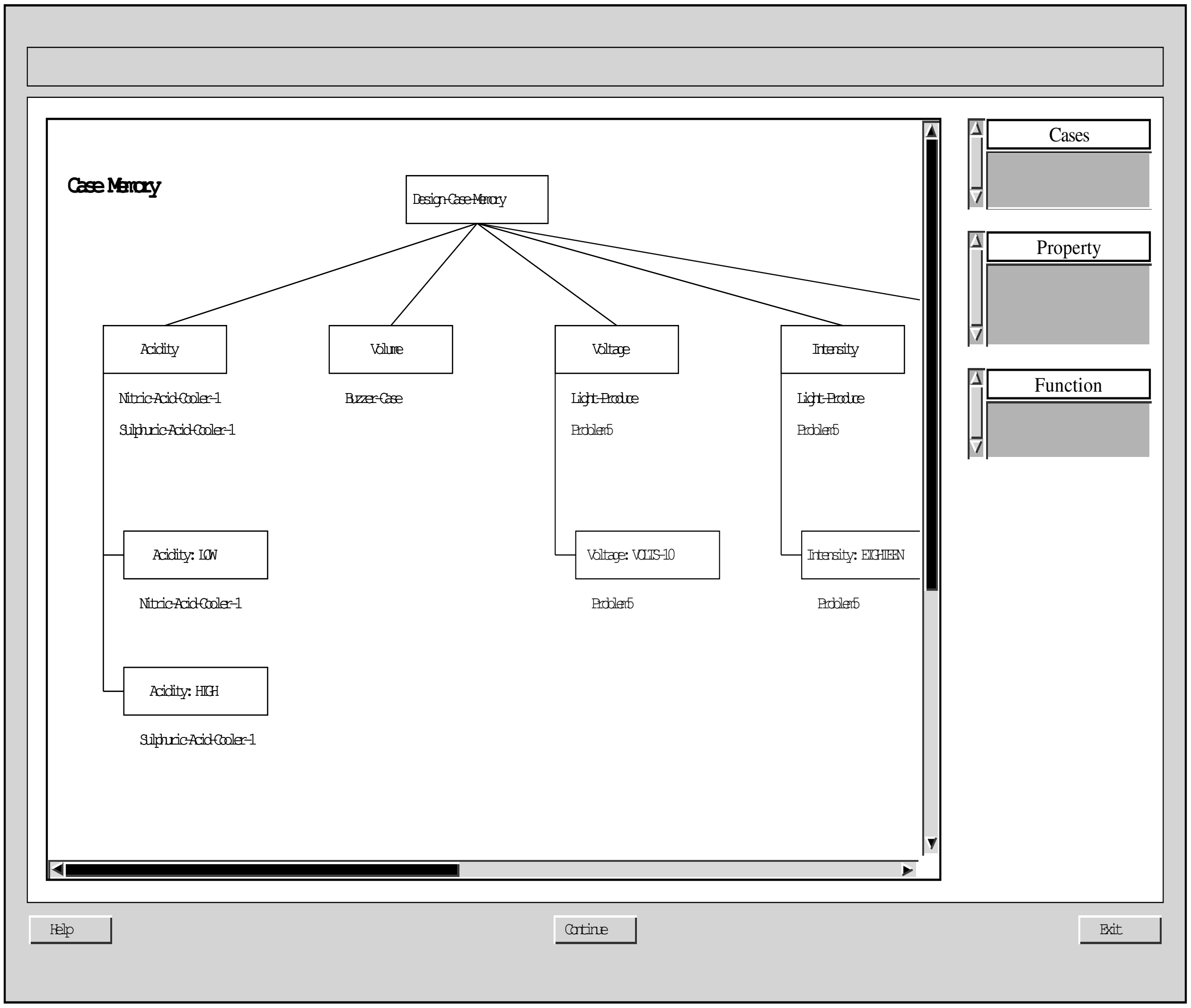


Figure B.35: Modified Memory of Design cases

Figure B.36 shows the complete decomposition of the *Design* task into four subtasks with their respective inputs and outputs. The user double-clicks on the *Exit* button to end CANAH-CHAB's session.

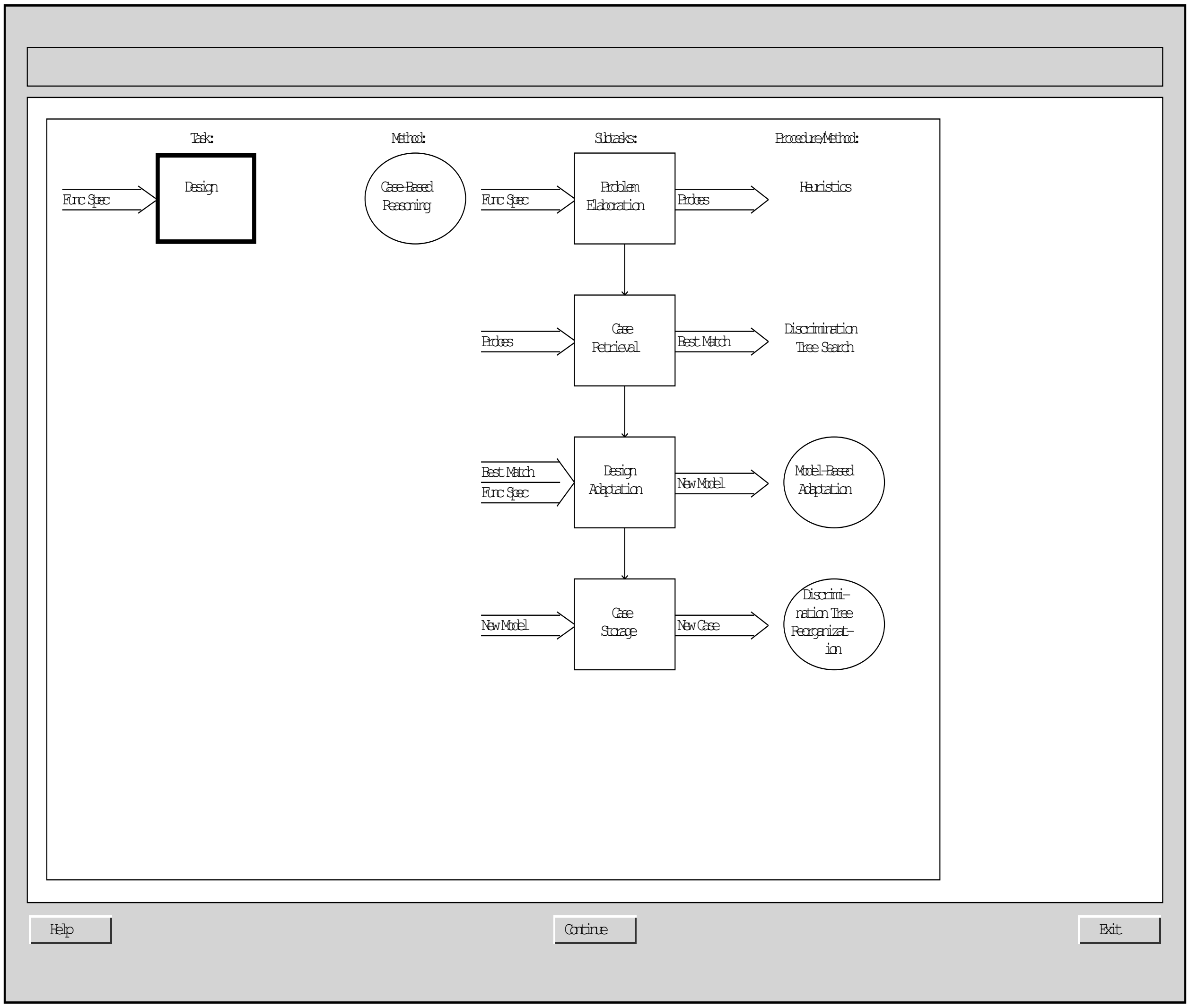


Figure B.36: Complete Design Task

Bibliography

[Allemang 1990] D. Allemang. Understanding Programs as Devices. PhD Thesis. The Ohio State University. 1990.

[Barber *et al.* 1992] J. Barber, S. Bhatta, A. Goel, M. Jacobsen, M. Pearce, L. Penberthy, M. Shankar, R. Simpson and E. Stroulia. AskJef: Integration of Case-Based and Multimedia Technologies for Interface Design Support. *Proceedings of the Second International Conference on Artificial Intelligence in Design*, pp. 457-476. Kluwer Academic. 1992.

[Baudin *et al* 1993] C. Baudin, S. Kedar, J. Gevins and V. Baya. Question-Based Acquisition of Conceptual Indices for Multimedia Design Documentation. *Eleventh National Conference on AI.* Washington D.C. 1993.

[Bhatta and Goel 1992] S. Bhatta and A. Goel. Use of Mental Models for Constraining Index Learning in Experience-Based Design. *Proceedings of the AAAI workshop on Constraining Learning with Prior Knowledge*, pp. 1-10. San Jose, CA. 1992

[Bhatta and Goel 1993] S. Bhatta and A. Goel. Model-Based Learning of Structural Indices to Design Cases. *Proceedings of the IJCAI workshop on "Reuse of Designs: An Interdisciplinary Cognitive Approach"*, pp. A1-A13. Chambery, Savoie, France. 1993.

[Bhatta, Goel and Prabhakar 1994] S. Bhatta, A. Goel and S. Prabhakar. Innovation in Analogical Design: A Model-Based Approach. To appear in *Proc. of the Third International Conference on AI in Design (AID'94),* Lausanne, Switzerland. 1994.

[Brown, Burton and de Kleer 1982] J. S. Brown, R. Burton and J. de Kleer. Pedagogical Natural Language and Knowledge Engineering Techniques in SOPHIE I, II, III. In *Intelligent Tutoring Systems*, S. Derek and J. S. Brown, (Ed). New York: Academic Press. 1982.

[Brown and Burton 1986] J. S. Brown and R. Burton. Reactive Learning Environments For Teaching Electronic Troubleshooting. In *Advances in Man-Machine Systems Research.* W. Rouse. (Ed). JAI PressInc, Greenwich, CT. Vol. 3. 1986.

[Bylander and Chandrasekaran 1985] T. Bylander and B. Chandrasekaran. Understanding Behavior Using Consolidation. *Proc. Ninth International Joint Conference on Artificial Intelligence*, pp. 450-454. 1985.

[Bylander 1991] T. Bylander. A Theory of Consolidation for Reasoning about Devices. *Man-Machine Studies.* Vol. 35, pp. 467-489. 1991

[Carbonell *et al* 1989] J. Carbonell, C. Knoblock and S. Minton. Prodigy: An Integrated Architecture for Planning and Learning. In *Architectures for Intelligence*, Lawrence Erlbaum. 1989.

[Chandrasekaran 1990] B. Chandrasekaran. Design Problem Solving: A Task Analysis. *AI Magazine*, pp. 59-71. Winter 1990.

[Chandrasekaran, Tanner and Josephson 1989] B. Chandrasekaran, M. Tanner, and J. Josephson. Explaining control strategies in problem solving. *IEEE Expert.* Vol. 4 No. 1, pp. 9-24. 1989.

[Clancey 1987] W. Clancey. Knowledge-Based Tutoring: The GUIDON Program. Cambridge, MA. MIT Press. 1987.

[Clancey 1993] W. Clancey. GUIDON-MANAGED Revisited: A Socio-Technical Systems Approach. *Journal of Artificial Intelligence In Education.* Vol. 4 No. 1, pp. 5-34. 1993.

[Davis 1979] R. Davis. Interactive Transfer of Expertise: Acquisition of New Inference Rules. *Artificial Intelligence.* Vol. 12, pp. 121-157. 1979.

[Domeshek and Kolodner 1991] E. Domeshek and J. Kolodner. Towards a Case-Based Aid for Conceptual Design. In *Int. Journal of Expert Systems Research and Applications.* Vol. 4 No. 2, pp. 201-220. 1991.

[Fischer *et al.* 1992] G. Fischer, J. Grudin, A. Lemke, R. McCall, J. Ostwald, B. Reeves and F. Shipman. Supporting Indirect Collaborative Design with Integrated Knowledge-Based Design Environment. *Human-Computer Interactions.* Vol. 7 No. 3, pp. 281-314. 1992.

[Forbus 1984] F. Forbus. Qualitative Process Theory. *Artificial Intelligence.* Vol. 24, pp. 85-168. 1984

[Goel 1989] A. Goel. *Integration of Case-Based Reasoning and Model-Based Reasoning for Adaptive Design Problem Solving.* Doctoral Dissertation. Department of Computer & Information Science. Ohio State University. 1989.

[Goel 1991a] A. Goel. A Model-based Approach to Case Adaptation. *Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society*, pp. 143-148. Lawrence Erlbaum Associates. 1991.

[Goel 1991b] A. Goel. Model Revision: A Theory of Incremental Model Learning. *Proceedings of the Eighth International Conference on Machine Learning*, pp. 605-609. Chicago. 1991.

[Goel 1992] A. Goel. Representation of Design Functions in Experience-Based Design. In *Intelligent Computer Aided Design*, pp. 283-308. D. Brown, M. Waldron and H. Yoshikawa (editors). North-Holland. 1992.

[Goel and Chandrasekaran 1989] A. Goel and B. Chandrasekaran. Functional Representation of Designs and Redesign Problem Solving. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pp. 1388-1394. Morgan Kaufmann Publishers. 1989.

[Goel and Chandrasekaran 1992] A. Goel and B. Chandrasekaran. Case-Based Design: A Task Analysis. In *Artificial Intelligence Approaches to Engineering Design, Volume II: Innovative Design*, pp. 165-184. Tong and D. Sriram (editors). Academic Press. 1992.

[Goel *et al.* 1993] A. Goel, M. Pearce, A. Malkawi and K. Liu. A Cross-Domain Experiment in Case-Based Design Support: ArchieTutor. *Proceedings of the AAAI Workshop on Case-Based Reasoning*, pp. 111-117. 1993.

[Goel and Prabhakar 1991] A. Goel and S. Prabhakar. A Control Architecture for Model-Based Redesign Problem Solving. In *Procs. of the IJCAI 1991 Workshop on AI in Design.* Sydney, Australia. 1991.

[Govindaraj 1987] T. Govindaraj. Qualitative Approximation Methodology for Modeling and Simulation of Large Dynamic Systems: Applications to a Marine Power Plant. *IEEE Transactions on Systems, Man and Cybernetics.* Vol. SMC-17 No. 6, pp. 937-955. 1987.

[Johnson 1993] K. Johnson. Exploiting a Functional Model of Problem Solving for Error Detection in Tutoring. PhD Thesis. The Ohio State University. 1993.

[Kedar *et al.* 1993] S. Kedar, C. Baudin, L. Birnbaum, R. Osgood and R. Bareiss. Ask How It Works: An Interactive Intelligent Manual for Devices. *INTERCHI'93.* 1993.

[Kolodner 1991] J. Kolodner. Improving Human Decision Making Through Case-Based Decision Aiding. In *AI Magazine.* Vol. 12 No. 2, pp. 52-68. 1991

[Kolodner 1993] J. Kolodner. *Case-Based Reasoning.* San Mateo, CA. Morgan Kaufmann Publishers. 1993.

[Laird, Rosenbloom and Newell 1986] J. Laird, P. Rosenbloom and A. Newell. Chunking in SOAR: The anatomy of a General Learning Mechanism. Kluwer Academic Publishers. Boston. 1986.

[Majumder, Fulton and Shilling 1990] D. Majumder, R. Fulton and J. Shilling. Designer In The ICAD Environment : The information Perspective. In *proceedings of the 1990 ASME International Computers in Engineering Conference and Exposition.* C. Born, W. Rasdorf and R. Fulton (Eds). 1990.

[Merrill, Reiser, Beekelaar and Hamid 1992] D. Merrill, B. Reiser, R. Beekelaar, and A. Hamid. Making Processes Visible: Scaffolding Learning with Reasoning-Congruent Representations. *Proceedings of the Second International Conference on Intelligent Tutoring Systems*, pp. 103-110. Springer-Verlag. 1992.

[Mostow 1989] J. Mostow. Design by Derivational Analogy: Issues in the Automated Replay of Design Plans. *Artificial Intelligence.* 1989.

[Myers 1990] B. Myers, D. Giuse, R. Dannenberg, B. Vander, D. Kosbie, P. Marchal, E. Pervin, A. Mickish and J. Kolojejchick. The Garnet Toolkit Reference Manuals: Support for Highly-Interactive, Graphical User Interfaces in Lisp. *Technical report CMU-CS-90-117-R.* School of Computer Science. Carnegie Mellon University. June 1991.

[Myers and Zanden 1992] B. Myers and B. Zanden. Environment for rapidly creating interactive design tools. *Visual Computer.* Vol. 8, pp. 94-116. 1992.

[Pearce *et al.* 1992] M. Pearce, A. Goel, J. Kolodner, C. Zimring, L. Sentosa and R. Billington. Case-Based Design Support: A Case Study in Architectural Design. In *IEEE Expert.* Vol. 7 No. 5, pp. 14-20. 1992.

[Recker and Pirolli 1993] M. Recker and P. Pirolli. Modeling Individual Differences in Students' Learning Strategies. *Journal of the Learning Sciences.* In Press.

[Richer and Clancey 1985] M. Richer and W. J. Clancey. GUIDON-WATCH: A graphic interface for viewing a knowledge-based system. *IEEE Computer Graphics and Applications.* Vol. 5 No. 11, pp. 51-64. 1985

[Rittel 1972] H. Rittel. On the Planning Crisis: System Analysis of the First and Second Generations. *Bedriftsokonomen.* Vol. 8, pp. 390-396. 1972.

[Schank 1991] R. Schank. Case-Based Teaching: Four experiences in Educational Software Design. Technical Report. Northwestern University. Institute for the learning Sciences. 1991.

[Shortliffe 1976] E. Shortliffe. Computer-Based Medical Consultation: MYCIN. *American Elsevier.* New York. 1976.

[Schöen 1987] D. Schöen. *Educating the Reflective Practitioner.* Jossey-Bass Publishers. 1987.

[Sembugamoorthy and Chandrasekaran 1986] V. Sembugamoorthy and B. Chandrasekaran. Functional representation of devices and Compilation of Diagnostic Problem Solving Systems. In *Experience, Memory and Reasoning*, J. Kolodner and C. Riesbeck (editors). Hillsdale, New Jersey: Erlbaum, pp. 47-73. 1986.

[Stasko 1990] J. Stasko. TANGO: A Framework and System for Algorithm Animation. *IEEE Computer.* Vol. 23 No. 9, pp. 27-39. 1990.

[Stasko and Patterson 1992] J. Stasko and C. Patterson. Understanding and Characterizing Software Visualization Systems. *IEEE Visual Languages Workshop'92*, pp. 3-10. Seattle, WA. 1992.

[Stroulia and Goel 1993] E. Stroulia and A. Goel. Using SBF Models of Problem Solving for Reflective Learning. *Proceedings of the IJCAI Workshop on Explanation and Problem Solving*, pp. 33-42. 1993.

[Stroulia and Goel 1994] E. Stroulia and A. Goel. Functional Representation and Reasoning for Reflective Systems. *Applied Artificial Intelligence.* To appear 1994.

[Vasandani and Govindaraj 1994] V. Vasandani and T. Govindaraj. Knowledge structures for a computer-based training aid for troubleshooting a complex system. In *The Use of Computer Models for Explication, Analysis and Experiential Learning*, D. Towne (editor). NATO ASI Series F, Programme AET. Springer-Verlag. To appear. 1994.

[Weintraub 1991] M. Weintraub. An Explanation-Based Approach to Assigning Credit. PhD Thesis. The Ohio State University. 1991.

[Wilensky 1984] R. Wilensky. Meta-Planning: Representing and Using Knowledge About Planning in Problem Solving and Natural Language Understanding. In Cognitive Science. Vol. 5, pp. 197-234. 1984

[Winston 1992] P. Winston. *Artificial Intelligence (3rd ed.)*. Addison-Wesley. 1992.