Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

H. Takeda, T. Tomiyama, H. Yoshikawa, P.J. Veerkamp

Modeling design processes

Computer Science/Department of Interactive Systems Report CS-R9059 October

Modeling Design Processes

Hideaki Takeda, Tetsuo Tomiyama, and Hiroyuki Yoshikawa
Department of Precision Machinery Engineering
Faculty of Engineering, The University of Tokyo
Hongo 7-3-1, Bunkyo-ku, Tokyo 113, Japan

Paul Veerkamp
Department of Interactive Systems
Centre for Mathematics and Computer Science
P.O.Box 4079, 1009 AB Amsterdam, The Netherlands

This article aims at building a computable design process model which is a prerequisite for realizing intelligent CAD systems. First, we discuss design processes seen from a theoretical point of view, i.e. we introduce General Design Theory from which a descriptive model of design processes is derived. In this model, the concept of metamodels plays a crucial role to describe the evolutionary nature of design. Second, we examine design processes from a cognitive viewpoint. We show a cognitive design process model obtained by observing design processes using a protocol analysis method. Then, we discuss a computable model that can explain most parts of the cognitive model and that also can interpret the descriptive model. In the computable model, a design process is regarded as an iterative logical process realized by abduction, deduction, and circumscription. We implemented a design simulator that can trace design processes in which design specifications and design solutions are progressively revised as the design proceeds.

CR Categories and Subject Descriptors: D.3, F.4.1, J.6.
Key Words & Phrases: intelligent CAD, design theory, design process model, meta-model, abduction, deduction, circumscription.

1. Introduction

One of the major problems in developing so-called intelligent CAD (Computer Aided Design) systems [19] is the representation of design knowledge which is twofold; the representation of design objects and design processes. We believe that intelligent CAD systems will be fully realized only when these two types of representations are integrated together. Progress is made in the former as can be observed for instance in geometric modeling, while there is almost no significant result in the latter. This implies that we need a design theory to formalize design processes. Some design theories have been already proposed, but they vary on their aims, domains, and so on.

According to Finger and Dixon [8], design process models can be categorized into a *descriptive* model which explains how design is done, a *cognitive* model which explains the designer's behavior, a *prescriptive* model which shows how design must be done, and a *computable* model which expresses a method by which a computer may accomplish a task. A design theory for intelligent CAD is not useful merely to be *descriptive* or *cognitive,* but also to be *computable*. We need a general model of design to clarify and define what design is from a

Report CS-R9059
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

theoretical point of view, which is a role of the descriptive model. But descriptive models are not necessarily helpful directly to derive either the architecture of intelligent CAD or the knowledge representation for intelligent CAD. For this purpose, we need a computable design process model that should coincide, at least to some extent, with a cognitive model which explains actual design activities.

In this article we discuss our attempt to establish a logical computable framework to represent design processes. This framework is not proposed merely for computability. It has foundation on discussions about our descriptive and cognitive models.

We start this article with the descriptive model. We present our descriptive theory called *General Design Theory*. It is a mathematical formulation of design processes and explains how design is conceptually preformed in terms of knowledge manipulation. Next, we show a more concrete descriptive model called *evolutionary design process model*. It is based on General Design Theory and shows how the design object is manipulated in the design process in an intelligent CAD environment. We then analyze design processes based on experimental results. We conducted design experiments to extract the features of design processes which a cognitive design model is derived from.

These discussions lead to a logical formalization of design processes in terms of a design process model which is not only coherent with the descriptive and cognitive models but also expected to be computable. In this model, a design process is regarded as a logical process in which both the theory (i.e. axioms) and the goal are revised gradually as the design proceeds, using deduction, abduction, and circumscription. Abduction is used to expand the designer's thought, deduction is used when the designer wants to get all obtainable facts from currently available design knowledge, and design object descriptions, and circumscription is applied to solve an inconsistency found during deductive reasoning. We illustrate a *design simulator* to demonstrate that this model is computable and also appropriate to represent design processes. An example is shown, in which, given design knowledge and specifications, the design simulator logically simulates a design process based on protocol data obtained by a real design experiment. In the last chapter, we discuss the relationship between the descriptive, cognitive, and computable design process models.

2. General Design Theory

We have developed General Design Theory (GDT) [20, 30] and its major achievements are a mathematical formulation of the design process and a justification of knowledge representation techniques in a certain situation. GDT is a descriptive model that tries to explain how design is conceptually performed in terms of knowledge manipulation. In GDT, a design process is regarded as a mapping from the function space onto the attribute space both of which are defined on the entity concept set. Based on axiomatic set theory, we can mathematically derive interesting theorems which can well explain a design process. (Interested readers may refer to [20, 30] for more precise discussions. In this paper we present the theorems without proof to limit the paper's size.)

2.1. Design in the ideal knowledge

GDT deals with concepts that exist only in our mental recognition. In this sense, GDT is an abstract theory about knowledge. We make a distinction between an *entity* and an *entity concept*. The former is a concrete existing thing, while the latter is its abstract mental impression conceived by a human being. An entity concept might be associated with its properties, such as color, size, function, place, etc. These properties are called abstract concepts and include attributes.

Axiom 1 (Axiom of recognition)

Any entity can be recognized or described by attributes and/or other abstract concepts.

Axiom 2 (Axiom of correspondence)

The entity set S′ *and the set of entity concepts (ideal)* S *have one-to-one correspondence.*

Axiom 3 (Axiom of operation)

The set of abstract concepts is a topology of the set of entity concepts.

Axiom 2 guarantees the existence of a superman who knows everything. In other words, it defines an ideal, ultimate state at which our knowledge should aim. Axiom 3 signifies that it is possible to operate abstract concepts logically, as if they were just ordinary mathematical sets. Accordingly, we get set operations, such as intersection, union, negation, etc.

Then, we can introduce *ideal knowledge* which knows all of the elements of the entity set and that can describe each element by abstract concepts without ambiguity. Theorem 1 describes mathematically this situation.

Theorem 1

The ideal knowledge is a Hausdorff space.

In GDT, a design specification is given as an abstract concept that the design solution must belong to. Thus, the specifications can be given by describing an entity with only abstract concepts (e.g. functionally). The function space is the entity concept set with a topology of functions, while the attribute space is the one with a topology of attributes. Therefore, a design specification is a point in the function space, while a design solution is a point in the attribute space. The most significant result of having the ideal knowledge which can be further proven from those three axioms is that design as a mapping from the function space to the attribute space successfully terminates when the specifications are described.

Theorem 2

In the ideal knowledge, the design solution is obtained immediately after the specifications are described.

Since we perfectly know everything in the ideal knowledge, when we complete describing the specifications these converge into a point in the function space. Because the function space and the attribute space are built on the same entity concept set, this point (i.e. an entity concept) can also be considered in the attribute space. The ideal knowledge is perfect also to describe an entity concept in the attribute space. Thus, the design solution will be fully described by attributes. This means that design in the ideal knowledge is a mapping process from the function space to the attribute space (see Fig. 1).

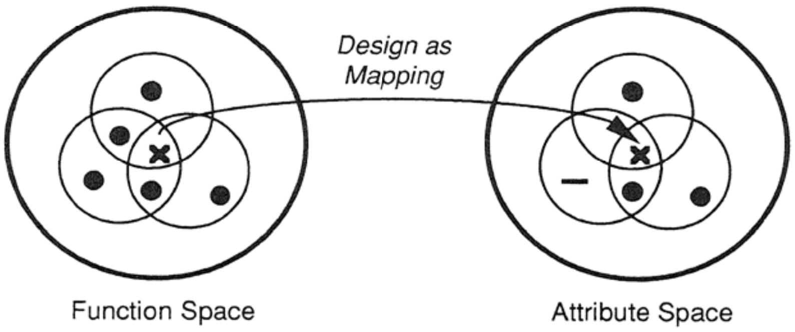


Fig. 1. Design process in the ideal knowledge

2.2. Design in the real knowledge

The situation in the ideal knowledge is not the case in the real design and we need to take the following characteristics into consideration:

(1) Design is not a simple mapping process but rather a stepwise refinement process where the designer seeks the solution satisfying constraints.

(2) The concept of function is difficult to formalize objectively, because it includes sense of value which can be different from person to person. Instead, we use *behavior* to deal with function.

(3) The ideal knowledge does not take physical constraints into consideration and it may produce design solutions such as perpetual machines.

These restrictions are considered in the *real knowledge* where design is regarded as a process that the designer builds the goal and tries to satisfy the specifications without violating physical constraints. In order to formalize the real knowledge, we first define a physical law as a description about the relationship between physical quantities of entities and the field. The concept of physical laws is one of the abstract concepts formed when one looks at a physical phenomenon as manifestation of physical laws. Physical laws constrain entities in the real world; in other words any feasible entity must be explicable by physical rules. This fact can be proven as a theorem.

Theorem 3

The set of physical law concepts is a base of the attribute concept topology of the set of (feasible) entity concepts.

This theorem states that attributes can be *measured* by using physical laws. An interesting fact about the real knowledge is that we can prove finiteness or boundedness of our knowledge by having the following hypothesis.

Hypothesis

There exist finite subcoverings for any coverings of the set of feasible entity concepts made of sets chosen from the set of physical law concepts.

Basically, this hypothesis says that a feasible entity is explicable not by an infinite number but a finite number (as small as possible) of physical laws. From this hypothesis, we can prove an interesting theorem which explains that an attribute has a value, if it is possible at all to measure the distance. Mathematically speaking, we can also prove the following theorems.

Theorem 4

The real knowledge is a compact Hausdorff space.

Theorem 5

In the real knowledge, if we can produce a directed subsequence from the given design specifications, this subsequence converges to a single point.

Theorem 5 indicates that in the real knowledge a design process can be regarded as a convergence process, if we can pick up some (but not necessarily all) specifications that seem to yield meaningful solutions.

2.3. Evolutionary design process model

The next step is to formalize design processes in the real knowledge. For this purpose, the concept of *metamodels* is formally introduced, where a metamodel is a finite set of attributes and the metamodel set is the set of all metamodels. Such a metamodel can be evolved; by to *evolve*, we mean to increase the number of attribute concepts.

Theorem 6

If we evolve a metamodel, we get an entity concept as the limit of evolution.

Theorem 6 is a corollary of Theorem 5 and it does not guarantee that we obtain a design solution as the result of this evolution. However, if we restrict ourselves to consider function only in terms of behavior, the following theorem can be proven.

Theorem 7

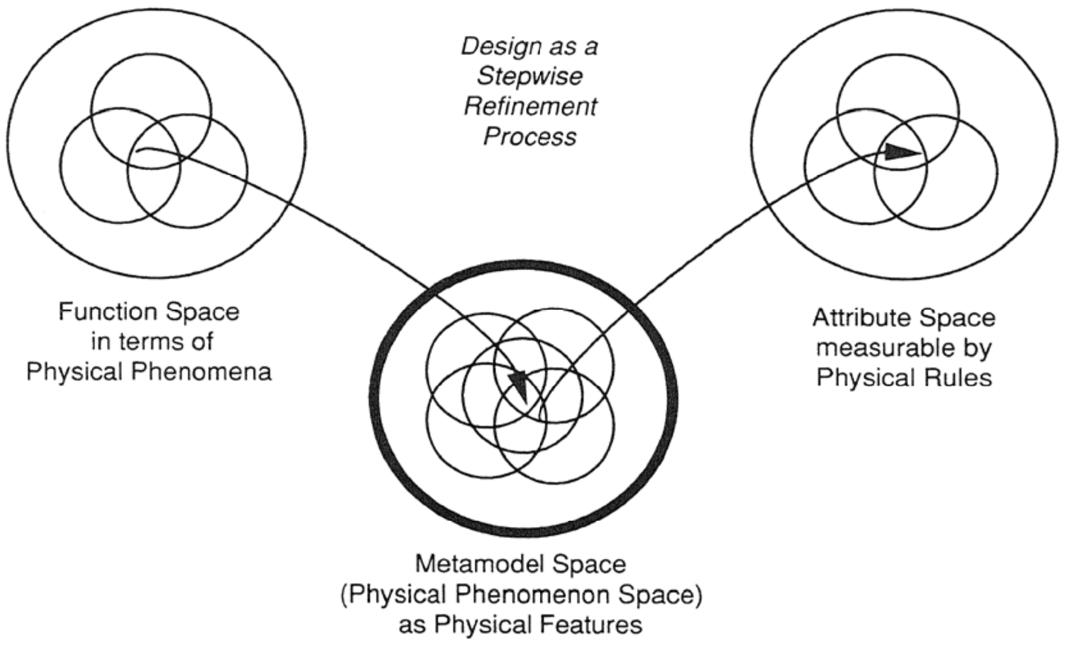
If we choose concepts that can be explained by physical law concepts as the metamodel, we can describe the design specifications by the topology of metamodel, and there exists the design solution which is an element of this metamodel.

Theorem 7 guarantees that we are able to design as far as specifications are given in terms of (physical) behaviors and solutions are described in terms of attributes that can be measured by physical laws. Furthermore, solutions contain only those which can be realized physically; in other words we are not allowed to consider objects that contradict to physical laws. Fig. 2 depicts a design process in the real knowledge in which we design, in fact, physical behaviors of the design object.

At the same time, Theorem 6 indicates that a design process is a stepwise, transformation process and solutions are obtained in a gradual refinement manner, because the metamodel can be evolved only by increasing the number of attributes. In the ideal knowledge, design is a direct mapping process from the function space to the attribute space, while in the real knowledge, design is a stepwise, evolutionary transformation process. This will be elaborated in the next chapter.

3. Descriptive Design Process Model

In this chapter we present a descriptive design process model which shows design as a gradual transformation from the function space onto the attribute space. This model is used to build a framework which directs the design process to be implemented in an intelligent CAD system.



Design as a Stepwise Refinement Process
Function Space in terms of Physical Phenomena
Attribute Space measurable by Physical Rules
Metamodel Space (Physical Phenomenon Space) as Physical Features

Fig. 2. Design process in the real knowledge

Knowledge about the design process is embedded in the framework. Thus, the system is always informed about the current stage of the design process and the current state of the design object [1]. Therefore, the descriptive design process model is described by using a more system-oriented terminology.

The knowledge about how to perform the design process and which tools are applicable at a certain instant is denoted by means of *scenarios*. These scenarios represent design schemes which describe what kind of actions must be carried out at a specific phase of the design process [9, 11]. For each phase of the design process there is a different set of scenarios.

The descriptive model is used as a basis to develop an intelligent CAD system, while the computable model, discussed later, simulates the reasoning aspect of design. Both of them are implementable on a computer; however, the former is meant to be a guide to discuss the architecture of an intelligent CAD system. The latter is more interesting for illustrating that design as a reasoning process can be performed by a computer.

3.1. From specification to solution

We use GDT as a basis for giving a descriptive design process model. The basic ideas behind the descriptive design process model are derived from the evolutionary design process model as follows:

- From the given functional specifications a candidate for the design solution is selected and refined in a stepwise manner until a complete solution is obtained, rather than by trying to get the solution directly from the specifications. The latter is not possible in a non-trivial design problem, since it involves a very complex object with a multitude of parts.

- The design process is regarded as an evolutionary process which transfers the model of the design object from one state to another, gradually obtaining a more detailed description. The number of attributes grows as the design process proceeds and a growing number of the functional specifications is met.
- To evaluate the current state of the design object model, various interpretations of the design object model need to be derived in order to see whether the object satisfies the specifications or not.

We call those interpretations of the design object model *contexts* and they can be regarded as interpretations of the design object observed from certain points of view. Contexts allow a designer to model the current state of the design object in a certain environment, i.e. they represent an *aspect model*. More information about the design object is obtained through these contexts and hence the number of attribute grows. Contexts are created by means of *scenarios* which contain design knowledge and data necessary to build an aspect model. Scenarios perform the reasoning about a context and they lead the dialogue with the designer.

3.2. Stepwise refinement of metamodel

Considering GDT, a designer starts with the functional specification of a design object and continues the design process until a design solution is obtained. During this process the design object model is refined in a stepwise manner. The central description of the design object is called a *metamodel*. It is called metamodel, because it is used during the design process as a central model from which aspect models are derived.

Note that, although here the definition of the metamodel is different from that of the previous chapter, the principle is the same. A metamodel in GDT is a set of attributes that include descriptions concerning physical phenomena. Here we show the use of the metamodel to integrate several aspect models.

A metamodel is a description of the design object which is independent of a context. It contains all entities the design object is composed of, and it includes the relationships and dependencies among these entities. Data which is used in one of the contexts derived from the metamodel is stored apart from the metamodel. For instance, geometric data of the design object is not to be found in the metamodel, but in a geometric aspect model. An example of a metamodel is shown in Fig. 3. It contains four entities and several relationships among these entities. Note that there are relationships which take effect on only a single entity. Other relationships take effect on several entities. During the course of design, the metamodel is refined in a stepwise manner, gradually obtaining a more detailed description.

The stepwise refinement process shown in Fig. 4 behaves as follows: at a certain stage of the design process the metamodel M_{i-1} is the current, incomplete description of the design object. In order to get a more detailed description an aspect model is derived from the metamodel. Through this aspect model some new information about the design object is obtained. After this refinement the new information from the aspect model is merged into the metamodel M_{i-1}. If the merge is successful, i.e. the new information is consistent with the current M_{i-1}, then the result of the merge is a new state of the metamodel, M_i. This process continues, obtaining M_{i+1}, etc., until the design object model becomes finally a complete and satisfactory description of the desired

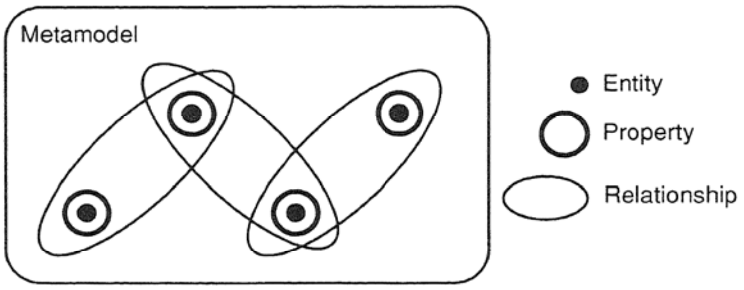


Fig. 3. Example of metamodel

artifact. We stress that here 'complete' has the meaning of satisfying all initial requirements. As a matter of fact a design can never be complete; there is always something which can be improved, or which can be made cheaper. In this context complete has a rather subjective meaning.

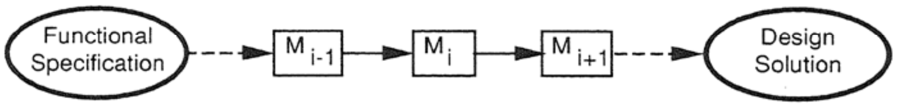
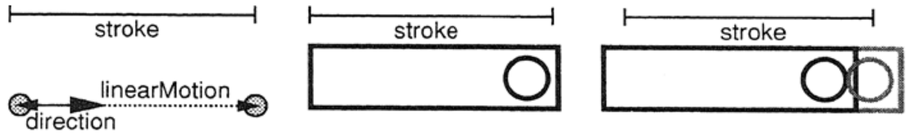


Fig. 4. Stepwise refinement of the metamodel

In Fig. 5 an example of this process is depicted. It shows several stages of the design of an linear motion mechanism. In Fig. 5(a) the state of design is at the conceptual design phase. The metamodel consists of an abstract anatomical description of the design object. Some states later the design is arrived at the fundamental design phase (see Fig. 5(b)). The metamodel is a concrete anatomical description of the design object without detail, i.e there are some inconsistencies between the geometric and the kinematic representations. The stroke achieved by the geometric representation is not the desired stroke. The detail is present in Fig. 5(c) when the design is at the detailed phase. Here, the metamodel consists of an exact anatomical description which is almost complete. Inconsistencies caused by results from different aspect models are removed. A more elaborated version of this example can be found in [25, 28].



a. Conceptual Phase b. Fundamental Phase c. Detailed Phase

Fig. 5. Example of stepwise refinement

3.3. Multiple representations

The design process as described above deals with the ideal situation in which the stepwise refinement process is a linear process from functional specifications straight to the design solution. It can be regarded as a sketch of the design process in retrospect. In practice, it is merely a process of trial and error, rather than the straightforward process shown in Fig. 4. The designer might not be satisfied with a certain state of the design and wants to redo it from a certain point. But he keeps in mind the things which were useful and which were not, and the redesign will therefore be more efficient. In another occasion the designer might like to regard the design object from different points of view at the same time, i.e. he wants to create multiple aspect models concurrently in order to compare the outcome from different experts.

During the course of the design, an occasion frequently occurs that the designer is not satisfied with the current state of design. Instead of redesigning everything from scratch, the designer wants to preserve part of the results. The designer restarts the design from a previous design state which still met his demands. The implication for the design process model is that it must be possible to withdraw the current metamodel and perform some backtracking to a previous one. In consequence, each individual state of the metamodel must be maintained as the design proceeds. On top of that, when the design is continued from a prior metamodel, it must be prevented from taking the direction which led to the unwanted result. Thus not only the metamodel states, but also the design process history must be maintained.

In Fig. 6 an example of the backtracking process is shown. For some reasons, the metamodel M_j does not fulfill the designer's requirements, so he decides to redesign from a prior state. In this case, he backtracks to the previous metamodel M_{i-1}. The design is restarted from this state taking a different direction. The design now proceeds to metamodel M_i, and so forth.

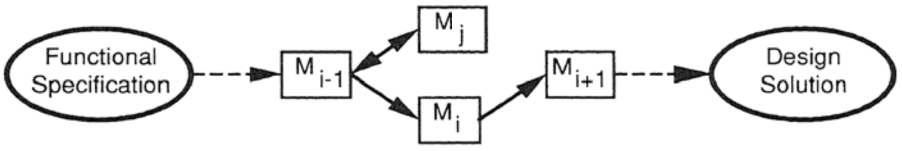


Fig. 6. Backtracking to a previous metamodel

3.4. Metamodel evolution scheme

The descriptive design process model, as we have introduced so far, gives a global outline of the design process, without going into details. We now focus on how to transfer from one state of the metamodel to the next; i.e. how do we perform a design step? Here we elaborate on the concept of a *context*. Through the creation of a context the designer provides an interpretation of metamodel in a certain environment, i.e. he generates an aspect model. For each design step there exists an associated scenario that creates a context which the design object is modeled in. The scenario contains the design knowledge which is applicable for its context, and it knows how to derive new data from the context and the current state of the metamodel. In a context new information about a design object is obtained. The current metamodel together with the new information form the next metamodel.

The design process is a continuous manipulation of the design object model. A certain aspect of the artifact is highlighted and some properties about the design object are changed. This highlighting is done by contexts. By applying a context, the attention is focussed on a specific part of the design object model. A context embodies a part of the metamodel with the addition of *aspect data*, design procedures, and design rules. Aspect data is information about the design object specific to a certain context, e.g. geometric information. It describes the context dependent properties of the design object. An example of a context is shown in Fig. 7. A context is an interpretation of the metamodel focusing on a certain aspect of the design object. It is used to derive new properties or to update uncertain or unknown properties about the metamodel in order to get a more detailed description. The new properties about the design object, which are derived in a context, are merged with the original metamodel when the modeling is completed.

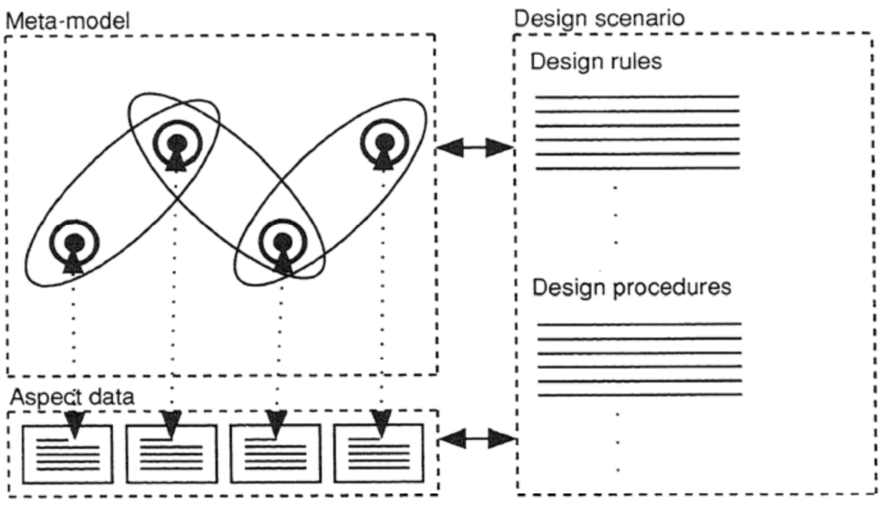


Fig. 7. Example of context

Here, we discuss the application of the *metamodel mechanism* [21, 24, 26]. A design step is performed by the execution of a design scenario. A scenario consists of design procedures and design rules which describe the procedural and declarative design knowledge respectively. The rules and procedures are applied to the metamodel and to the aspect data. In such a manner we create a context which consists of (a part of) the metamodel together with knowledge about the design object being valid in that particular context. The designer can interact with the context and change its contents. After the session the contents of the context being evaluated is mapped to the next state of the metamodel. The mapping is performed by the metamodel evolution mechanism presented below.

An example of such a context is a geometric representation of the design object. A part of the metamodel is collected and processed in a context with geometrical knowledge in order to generate an image of that part of the design object. In other words, there exists a scenario which creates a geometric aspect model. The aspect data of the model consist of the geometric attributes of the design object. The design procedures in the scenario describe geometrical functions, and the design rules describe geometric knowledge. Such a context describes the mapping of the metamodel onto a description suitable for the generation of a geometric model. Hence, the metamodel allows the designer to model the geometric aspects of the design object and therefore obtaining a more precise representation.

We can now give a definition of the mechanism which constitutes metamodel *evolution*, i.e. the metamodel mechanism. The metamodel mechanism is applied to perform a design step. It transfers the metamodel from its current state to the next, according to the stepwise refinement model. The mechanism is driven by scenarios. The designer selects a design scenario which is appropriate for the current state of the metamodel M_i. The scenario is executed in a context c_i and performs in dialogue with the designer some action on the context. The execution of a scenario continues as long as new information can be obtained in the context. The acquisition of new information is accomplished through the design procedures and the design rules in a scenario.

The contents of a context is evaluated, when the execution of its scenario is completed. The evaluation checks the context c_i for consistency with the metamodel M_i, i.e. there are no facts that contradict each other and all constraints over the design object model are met. The metamodel M_i is transferred to M_{i+1}, if the evaluation succeeds. In case of failure, all results of c_i are discarded and the process will restart from M_i (see Fig. 8). This backtracking is performed in dialogue with the designer, so that the next attempt can be made more successful.

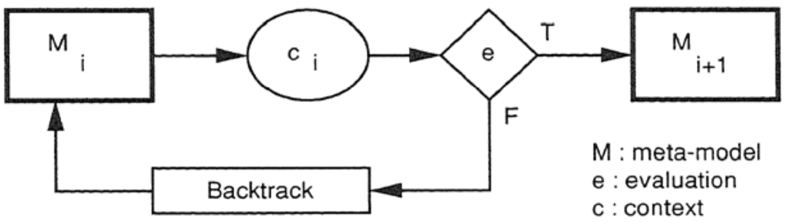


Fig. 8. Metamodel evolution

A consecutive application of the metamodel evolution mechanism enables the designer to perform a design job. Hence, the design process consists of the execution of a series of design scenarios. For each state of the metamodel, there exists an appropriate scenario. The descriptive design process model presented here, is derived from GDT, providing a framework which allows for design process representation. The metamodel mechanism is part of this framework and depicts the evolutionary nature of the design process, i.e. it represents a design step. The design process can therefore be modeled as a stepwise refinement process.

In this and the previous chapters, starting with discussing design theory from a theoretical viewpoint, we obtained a descriptive design process model. This model clarifies the representation of the design object that changes dynamically. We also suggested a way to apply this model to building intelligent CAD systems. It seems to be reasonably interesting to extend this descriptive model to a computable model that can clarify the reasoning process during the design process. But we do not only discuss design processes from a theoretical viewpoint. Since not only the design object but also the designer plays a crucial important role in design, we have to consider another viewpoint, i.e, a cognitive viewpoint, which is presented in the next chapter.

4. Design Experiment and a Cognitive Design Process Model

In this chapter, we discuss design processes from an experimental point of view. We used an experimental method called *design experiment* [17, 29, 31]. It is a kind of psychological experiment in which designers are asked to design an artifact from a given set of specifications. The whole designing session is video-taped and analyzed by protocol analysis methods. There are various differences between the experiment in psychology and one in design not only in task but also in environment and so on. We discussed how the experiment must be prepared to obtain useful results and performed several experiments based on this discussion [18]. This experimental approach is also studied by e.g. Ullman, Dietterich, and Stauffer [23]. and Goel and Pirolli [10].

One of our major results obtained from the experiments is a cognitive model of design processes when examining a design process from a point of view of problem solving [17]. This model is constructed from unit *design cycles* (see Fig. 9).

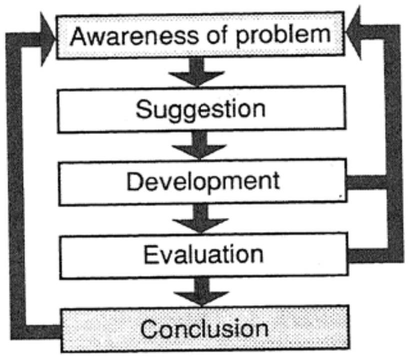


Fig. 9. Design Cycle

A design cycle consists of five subprocesses as follows.

(1) *Awareness of the problem* to pick up a problem by comparing the object under consideration and the specifications.

(2) *Suggestion* to suggest key concepts needed to solve the problem.

(3) *Development* to construct candidates for the problem from the key concepts using various types of design knowledge. When developing a candidate, if something is found something unsolved, it becomes a new problem that should be solved in another design cycle.

(4) *Evaluation* to evaluate candidates in various ways, such as structural computation, simulation of behavior, cost evaluation, etc. If a problem is found as a result of evaluation, it becomes a new problem to be solved in another design cycle.

(5) *Conclusion* to decide which candidate to adopt so as to modify the descriptions of the object.

Utterances in the protocol data are categorized into these subprocesses and then a design cycle is composed of these subprocesses. Basically a single design cycle solves one problem, and sometimes new problems that must be solved in other design cycles are arisen in the suggestion and evaluation subprocesses.

We can distinguish two levels in the design process when we consider the designer's mental activity. One is the *object level* on which the designer thinks about design objects themselves, e.g. what properties the design object has, how it behaves in a certain condition, and so on. The other is the *action level* on which the designer thinks about how to proceed her/his design, i.e, what she/he should do next. The designer seems to perform her/his design using these two types of thinking mutually. When looking at design cycles with this aspect, they also contain these two levels (see Fig. 10). The development and evaluation subprocesses are only present on the object level, while the awareness-of-problem, evaluation, and conclusion subprocesses are divided between these two levels. In the first two subprocesses, the designer has only to be concerned with the design object. On the other hand, in other subprocesses, the designer should not only take the design object but also the design process itself into consideration. In case the designer encounters a problem that cannot be solved due to e.g. lack of information, she/he may suspend solving it until she/he receives sufficient information or even try to abandon that particular design candidate. Action level decisions are needed for dealing with such occasions.

In this chapter we presented the cognitive model of design processes. In the next chapter, we will look at the reasoning aspect of the cognitive and descriptive models, which will result in the computable design process model.

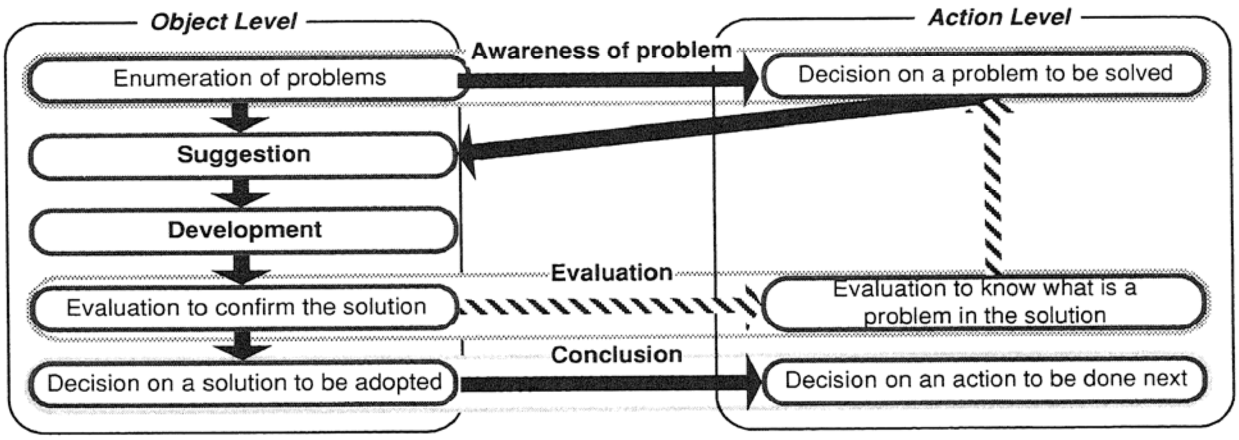

Object Level
Action Level
Awareness of problem
Enumeration of problems
Decision on a problem to be solved
Suggestion
Development
Evaluation
Evaluation to confirm the solution
Evaluation to know what is a problem in the solution
Conclusion
Decision on a solution to be adopted
Decision on an action to be done next

Fig. 10. Two levels in the design cycle

5. A Logical and Computable Model of the Design Process

We try to reconstruct the cognitive and descriptive models into a computable model in this chapter. Logic is a useful framework for this purpose, because it is developed from human thought, and it is also closely related to inference. Some results towards understanding design in logic are discussed in [3, 22] but they are concerned with design processes only at the conceptual level but neither at the decision making level nor the computable level. In other words, they end up with just showing a possibility of logical formalization.

5.1. The logical foundation for the computable model

In describing design processes logically, it is reasonable to assume the following simple model as a first step:

$$S \cup K \vdash Ds$$

where S, K and Ds are a set of formulae that denote the specifications, knowledge used in design, and the design solutions, respectively. The solutions are derived from the specifications and knowledge as the results of deduction. We took this approach in [17], and Forlog [4, 6] also used this approach. However, this approach has three difficulties in interpreting the whole of the design process.

First, design is not always performed with complete information. The above assumption requires complete descriptions for specifications. Although there are some domains such as VLSI design in which all the required specifications are given before the beginning of design, this cannot be applied to domains such as mechanical design where refinement of the specifications is a crucial part of the design process to obtain complete specifications from incomplete ones. This implies that refinement of specifications and designing of objects are performed mutually in design; i.e. refinement of specifications recall next designing processes and results of designing objects request to refine specifications. We employ not only deduction but also *abduction* [5] for the formalization of design processes to refine the specifications.

Second, knowledge in the above assumption is concerned with how to design objects, i.e. what the designer should consider the given specifications. A typical example is, ''if there is a specification S_1, then use a design object D_1''. Although it may be useful for routine design, it is not appropriate to more flexible and creative designs in which knowledge about object properties and behaviors plays an important role. We believe that knowledge on object properties and behaviors is more primary than knowledge about how to design, because a designer can design even when she/he faces a task to design a new product using such knowledge on object properties and behaviors.

The third problem, which is related to the second, is incompleteness and inconsistency of the knowledge. When the designed object does not satisfy the specifications, it should be regarded that the knowledge base was initially incomplete rather than inconsistent. Therefore, we need to deal with a method to make an incomplete knowledge base complete.

In this article, we regard design solutions and knowledge as the premise and the specification as the conclusion. Coyne and Treur also adopted this approach [3, 22]. Based on this approach, we formalize design processes as bidirectional and iterative processes as follows:

$$\begin{tikzcd} Ds \cup Ko \arrow[r, phantom, "\vdash"] \arrow[r, bend left] & P \arrow[l, bend left] \end{tikzcd}$$

where Ds is a set of logical formulae describing a design solution which the designer wants to obtain (we call it a design candidate because it is not yet a solution of the design), Ko is knowledge on object properties and behavior, and P are properties which the design candidate has. Required specifications are included in P. Given design knowledge Ko and the required properties P as the specifications, the designer tries to obtain a candidate which is expected to satisfy the specifications by using abduction. Then deduction is performed (i) to see what properties the candidate has and (ii) to see whether the candidate does not contradict with the given constraints including the specifications, because abduction merely proposes literally candidates. This results in that the obtained candidate should be detailed by getting its complete descriptions. If the candidate does not satisfy the specifications, the designer either tries an alternative candidate or modifies the design knowledge and the specifications. This recalls further abductive or deductive processes, and finally complete descriptions of the solution and the specifications are obtained. If there is no more way to explore, the design process terminates.

We can solve the first and the second problems in this formalism, but we still ought to deal with an incomplete knowledge base, which is the third problem. We introduce circumscription [14] for this problem. We assume that every piece of knowledge is valid only when it is used in certain situations; i.e. a contradiction occurs, not because knowledge as a set of logical formulae contains contradicting information, but because it is not used in an appropriate situation. However, it is not an easy task to explicitly describe applicable situations for every knowledge in advance. We can only identify the applicability of knowledge when detecting a contradiction. For a given set of logical formulae, circumscription can be used to compute exceptions that caused the contradiction and to formulate them explicitly as a new situation. By doing so, the original knowledge is modified and able to handle incompleteness.

5.2. The computable model

In the previous section, we discussed the idea of employing three different types of reasoning, i.e. deduction, abduction, and circumscription, for formalizing design processes. In this section, we show a computable model of design processes based on these three types of reasoning and we interpret the cognitive model in terms of the computable model.

Let us assume the following situation. We consider that a design process changes its state step by step. In each state, the following three types of descriptions hold. The first one is the description of the current design candidate, that consists of either the identifiers of the current design candidate or the properties which are necessary to identify the current candidate. In the following discussion it is denoted by Ds_c. The second one is the description of properties of the current design candidate, P_c. It consists of all kinds of properties of the current design candidate. The third one indicates the kind of knowledge that is available at the current state, Ko_c. These descriptions are kept consist by satisfying the following formula:

$$\mathsf{Ds}_c \cup \mathsf{Ko}_c \vdash \mathsf{P}_c.$$

Now we try to make a computable model by interpreting the cognitive model in the logical framework discussed above. First we concentrate on the object level.

The suggestion subprocess is a process where the designer tries to find a feasible candidate. This means that the purpose of this subprocess is to obtain Ds_c from P_c and Ko_c and it can be regarded as an abduction process. Peirce introduced abduction as an *ampliative reasoning*, i.e. it augments the knowledge. Since Peirce's abduction is considered logical reasoning from a consequent to an antecedent, it commits an error of *the fallacy of affirming the consequent*. In this sense, abduction is weak reasoning.

In our model, abduction is represented as follows;

Ds_2=abduction(Ds_1, K_1)

where

$Ds_1 \subset Ds_c$,
$K_1 \subset K_c$, and
$Ds_2 \cup K_1 \vdash Ds_1$.

Ds_1, which is a part of (or the whole of) the current design candidate, is replaced by Ds_2 using knowledge K_1. The new description of the current design candidate is

$Ds' = (Ds_c - Ds_1) \cup Ds_2$.

Ds' is expected to satisfy P_c with Ko_c, but there is no certainty because it is not the role of abduction to check the consistency of the entire object description in our model.

On the other hand, both the development and the evaluation process are regarded as deduction. In these subprocesses, the designer applies her/his knowledge to the candidates and obtains what is known at the current state. This is suitable for deduction, which is called *explicative reasoning* by Peirce. The difference between those two subprocesses lies in the kind of knowledge which is applied. The development subprocess uses knowledge to find out what properties the design object has, whereas the evaluation subprocess uses knowledge to compare those properties obtained in the development subprocess with expectations. The purpose of these two processes is to obtain P from Ds and Ko:

$Ds_c \cup Ko_c \vdash P'$.

P' may be different from P_c if and only if Ds_c or Ko_c changes. Ds_c changes as the result of abduction and Ko_c changes as the result of circumscription which is discussed below.

While the designer is developing or evaluating, she/he sometimes encounters a difficulty about the candidate and defines a new problem in order to solve it. It is a jump from a development or evaluation subprocess to an awareness-of-problem subprocess. We interpret it as circumscription.

In the logical framework the continuation of the design process may be disturbed by two reasons. One is lack of information, which will be dealt with by the meta-level reasoning. The other is the occurrence of a contradiction in the theory. As mentioned in the previous section, the contradiction does not happen because the knowledge contains false information but because it is incomplete; i.e. some pieces of knowledge are used in the situation where they must not. We can avoid this by defining exceptions for these pieces of knowledge using circumscription.

If a contradiction is detected in the theory, we gather formulae which cause the contradiction. Then we add literals that are composed of *abnormal* predicates to them and circumscribe these abnormal predicates with the theory. We obtain modified formulae in which their abnormal predicate are substituted by non-empty formulae. As the result, the contradiction is removed from the theory. For example, suppose the theory is

$\{ A \rightarrow B, A \rightarrow \neg C, \neg C \rightarrow \neg B, A \}$.

Since this theory is inconsistent, we add an abnormal predicate to each formula as follows;

$$\{ A \cap \neg ab_1 \rightarrow B,\ A \rightarrow \cap \neg ab_2 \rightarrow \neg C,\ \neg C \cap \neg ab_3 \rightarrow \neg B,\ A \}.$$

When we circumscribe ab_1, ab_2 and ab_3 with the theory, one of the results is a set of substitutions for abnormal predicates such as $ab_1 = \neg C$, $ab2 = \text{false}$ and $ab3 = \text{false}$. The formulae are then modified as

$$\{ A \cap C \rightarrow B,\ A \rightarrow \neg C,\ \neg C \rightarrow \neg B,\ A \}.$$

Circumscription modifies the knowledge currently used; it is expressed as follows:

$$Ko' = \text{circumscription}(Ko_{c},\ Ds_{c},\ P_{c}).$$

Since the formulae are modified, we cannot always derive the formulae that represent the required specifications from the current design candidate. Some new formulae must be added to the design candidate to satisfy the specifications. In this process, the contradiction creates a new problem, i.e. the awareness-of-process subprocess. In the example above, we can avoid the contradiction, but on the other hand we cannot conclude B, which is not satisfactory. One possibility to derive B is to state a new problem C to be solved first.

Now we can discuss the action level of the logical framework. One of the problems that occurs on the action level is how the awareness-of-problem subprocess becomes apparent. Although we discussed a jump from the development and the evaluation subprocess to the awareness-of-problem subprocess, we have not mentioned how to start a design cycle. This is one of the problems of logical reasoning, or rather a problem how to prepare the theory before starting to reason. Therefore, we introduce meta-level operations to deal with this problem.

Metal-level architectures for reasoning are suggested by many researchers. For example, Weyhrauch [27] proposed FOL which is a meta-level reasoning system based on first-order logic. In our approach, operations on building theories and on how to perform reasoning are introduced as meta-level operations including:

setting-up of Ds,
setting-up of P,
setting-up of Ko,
revision of Ds by abduction on P and Ko,
revision of P by deduction on Ds and Ko, and
revision Ko by circumscription on Ds, P and Ko.

Starting the design cycle is interpreted as execution of *setting-up of* P and *setting-up of* Ko operations. Similarly, we can interpret the suggestion, the development and the evaluation subprocess as the executions of these operations. Knowledge about how to design can be represented as a sequence of the operations on this level.

Reasoning on the action level is, therefore, to obtain a sequence of the operations on the object level. This reasoning is performed by using the status of the object level and knowledge about how to design. The status of the object level is determined by what kind of information is obtained on the object level and how this information is obtained. Knowledge about how to design, which is mentioned in the previous section, is only used on this level and includes procedural knowledge about how to proceed design, design strategy, and so on. It is possible to discuss reasoning on the action level in the same way we did for the object level. However, actually only deductive reasoning such as rule-based reasoning is needed because most of this type of knowledge is procedural.

We do not interpret the conclusion subprocess within the logical framework. It is a decision making process by considering all information obtained by other types of reasoning and therefore it seems a reasoning higher than those types of reasoning. We leave this subprocess unformalized.

Fig. 11 summaries the discussion, i.e. it shows the correspondence between the subprocesses and these types of reasoning in the logical framework.

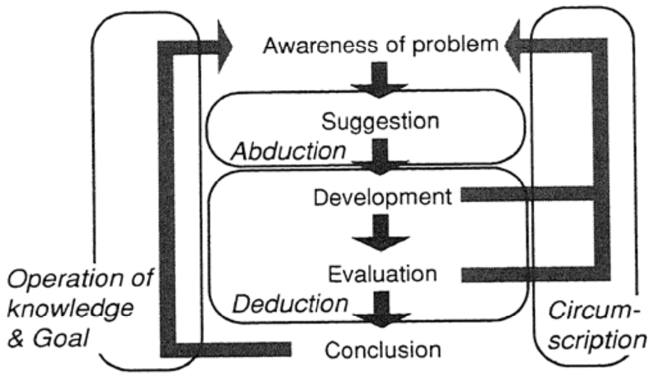


Fig. 11. Reasoning on the design cycle

6. The Design Simulator

We implemented a prototype of *design simulator* that realizes the inference discussed above. We call it the design simulator, because it is designed to track the design processes performed by designers.

6.1. The architecture

The design simulator consists of two main parts; i.e. the action level inference system and the object level inference. The object level inference consists of furthermore workspace Ds, P, and Ko, and three inference subsystems, i.e. deduction, abduction and circumscription subsystems (see Fig. 12).

The meta-level inference is performed by a rule-based system. Knowledge used on this level is about how to design, for example, knowledge about selecting a knowledge base and scheduling reasoning according to the condition of the object level. Results of this deduction is a sequence of operations on the object level which are described in the previous chapter.

The object level inference is performed by abduction, deduction, and circumscription to the workspace that modify the current state of Ds, P, and Ko. Ko contains knowledge as Horn clause, and P and Ds contain objects and their properties as atom formulae that have only a literal. The inference on the object level modifies the contents of Ds, P, or Ko and sometimes causes contradictions, which are reported to the meta-level as a condition of the object level.

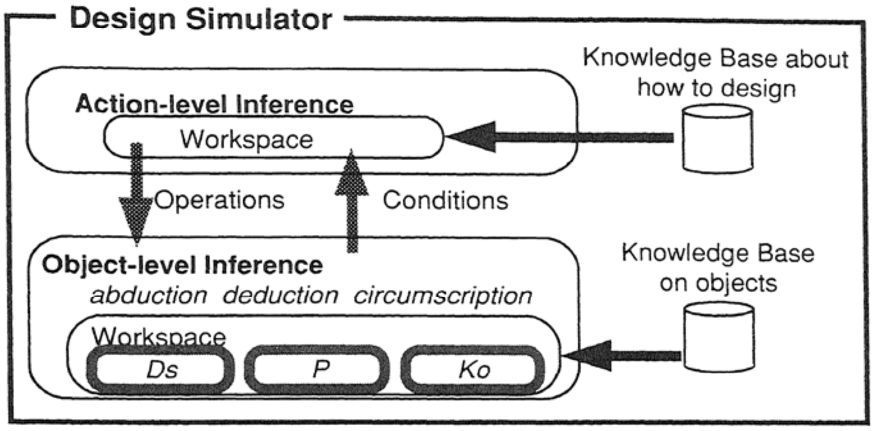


Fig. 12. The design simulator

Abductive reasoning and circumscription are realized as follows. There is some work on abductive reasoning, e.g. the RESIDUE system [7] and the work by Cox and Pietrzykowski [2]. They are based on the resolution principle in order to perform abductive reasoning. Abductive reasoning can be regarded as a process deriving a theory from formulae which denote knowledge and possible assumptions. Suppose G is a goal, K is a set of formulae representing knowledge, and A is a set of formulae representing all possible assumptions. We find a subset A to prove G by

$$K \cup A_1 \vdash G$$

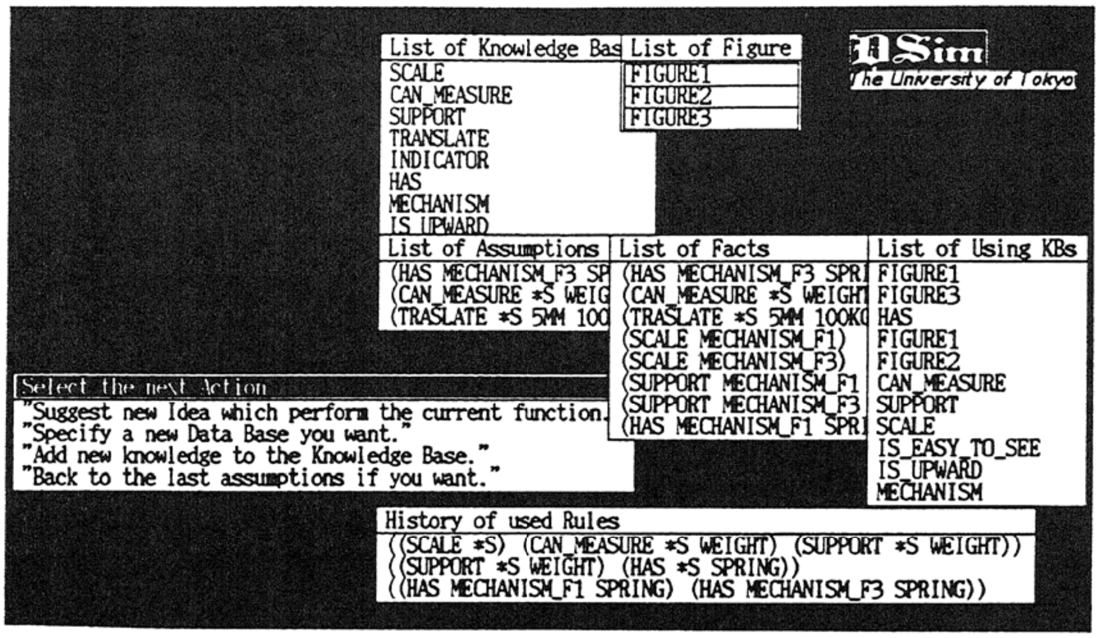
where $A_1 \subset A$. Then A_1 is obtained by abduction of G using K. This idea is shown by Poole [16] as the theory of default reasoning, but it can also be applicable to abductive reasoning. We implemented this abductive reasoning by modifying the algorithm of Prolog [13] to have two sets of clauses. One is the definition clause set which is K in the above formula and contains program clauses, and the other is the assumption clause set which is A and contains only unit clauses. Given a goal, the system resolves it until an empty clause is found as Prolog does, except that we first use clauses in the definition clause set. The clauses in the assumption clause set are used, if and only if there is no clause to be applied in the definition clauses. This process corresponds to finding A_1 from A in the above explanation. This matching strategy is provided to obtain *deeper* assumptions. Here, set of assumptions A_1 is *deeper* than set of assumptions A_2, if and only if $A_1 \cup K \vdash A_2$.

The circumscription system is implemented using the algorithm proposed by Nakagawa and Mori [15]. They developed an algorithm for computing circumscription [12] on clausal forms. Their basic idea is to use the technique of program transformation such as *unfolding* when eliminating variable predicates and abnormal predicates.

The design simulator accomplishes a reasoning process by using abduction, deduction and circumscription repeatedly. The order of them are specified by the action level, and it asks the users when there are some alternatives. This system is implemented in Allegro Common Lisp and X11 on Sun-4.

6.2. Example

We simulate a design process composed of protocol data obtained by a design experiment. Fig. 13 shows a snapshot of the design simulator solving this example. The task is to design a weighing scale and a part of the protocol data is shown in Fig. 14. To simulate design processes by the design simulator, we prepare data as follows. Observing the protocol data, we represent the designer's thought as a set of formulae shown in Fig. 15 (numbers in this figure correspond to ones in the protocol data in Fig. 14). We also translate the required specifications into logical form. We set the specifications to P, and the design simulator executes abduction and deduction when new knowledge is added and the candidate and its properties are revised.



```
List of Knowledge Bas
SCALE
CAN_MEASURE
SUPPORT
TRANSLATE
INDICATOR
HAS
MECHANISM
IS_UPWARD

List of Figure
FIGURE1
FIGURE2
FIGURE3

List of Assumptions
(HAS MECHANISM_F3 SP
(CAN_MEASURE *S WEIG
(TRASLATE *S 5MM 100

List of Facts
(HAS MECHANISM_F3 SPRI
(CAN_MEASURE *S WEIGHT
(TRASLATE *S 5MM 100KG
(SCALE MECHANISM_F1)
(SCALE MECHANISM_F3)
(SUPPORT MECHANISM_F1
(SUPPORT MECHANISM_F3
(HAS MECHANISM_F1 SPRI

List of Using KBs
FIGURE1
FIGURE3
HAS
FIGURE1
FIGURE2
CAN_MEASURE
SUPPORT
SCALE
IS_EASY_TO_SEE
IS_UPWARD
MECHANISM

Select the next Action
"Suggest new Idea which perform the current function.
"Specify a new Data Base you want."
"Add new knowledge to the Knowledge Base."
"Back to the last assumptions if you want."

History of used Rules
((SCALE *S) (CAN_MEASURE *S WEIGHT) (SUPPORT *S WEIGHT))
((SUPPORT *S WEIGHT) (HAS *S SPRING))
((HAS MECHANISM_F1 SPRING) (HAS MECHANISM_F3 SPRING))
```

Fig. 13. A snapshot of the design simulator

We illustrate a short session of the reasoning that corresponds to a part of the actual design session. At the beginning of the process, there exits only the specifications in P and nothing in Ds (see Fig. 17(a)). The formulae corresponding to the protocol (1)-(7) are added to Ko, and using by P and Ko, the content of Ds, which prepresents the current design candidate, is revised as in Fig. 17(b). And the content of P, which represents properties of the current candidate, is also revised as in Fig. 17(b). Since Ds contains the specifications, it satisfies the specifications. A contradiction is detected when adding knowledge of the protocol (8); the designer notices that an easy mechanism is needed for the machine in the human process. This contradiction occurs, because formula (2a) derives scale(*s) and formula (8a) derives not(scale(*s)). Then circumscription is executed, and as a result formula (2a) and (8a) are modified as shown in Fig. 16. The literal is_easy_mechanism added to formula (2a) is used in the next abduction, and the new formula is added to Ds. Fig. 17(c) shows the revised contents of Ds and P.

(1) What mechanism does a standard scale use?
(2) It measures the weight like this Fig. A).
...
(3) If we can use rack and pinion (Fig. B), we can measure the weight because the displacement is in proportion to the weight.
(4) Anyway, we think about the indicator first.
(5) When it translates 5mm of the displacement to the 100kg weight, the displacement is 0.05mm/kg.
(6) It is possible to realize it with Fig. B.
(7) If we don't mind the accuracy, it is possible by using many gears.
...
(8) But a standard scale must use a more simple mechanism.
...

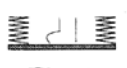


Fig. A

Fig. B

Fig. 14. The protocol data

Ko

(2a) scale(*s) ← can_measure(*s *w) ∧ support(*x *w)
(2b) support(*s *w) ← has(*s spring)
(2c) can_measure(*s *w) ← translate(*s *w *d) ∧ has(*s pinion)
(3a) translate(*s *w *d) ← is_in_proportion(*s *w *d)
(3b) is_in_proportion(*s *w *d) ← has(*s rack) ∧ has(*s pinion)
(6) translate(*s 100kg 5mm) ← has(*s indicator1) ∧ has(indicator1 many_gears)
(8a) not(weighing_machine(*s)) ← has(*s indicator1) ∧ not(is_easy_mechanism(indicator1))
(8b) not(is_easy_mechanism(indicator1)) ← has(indicator1 many_gears)

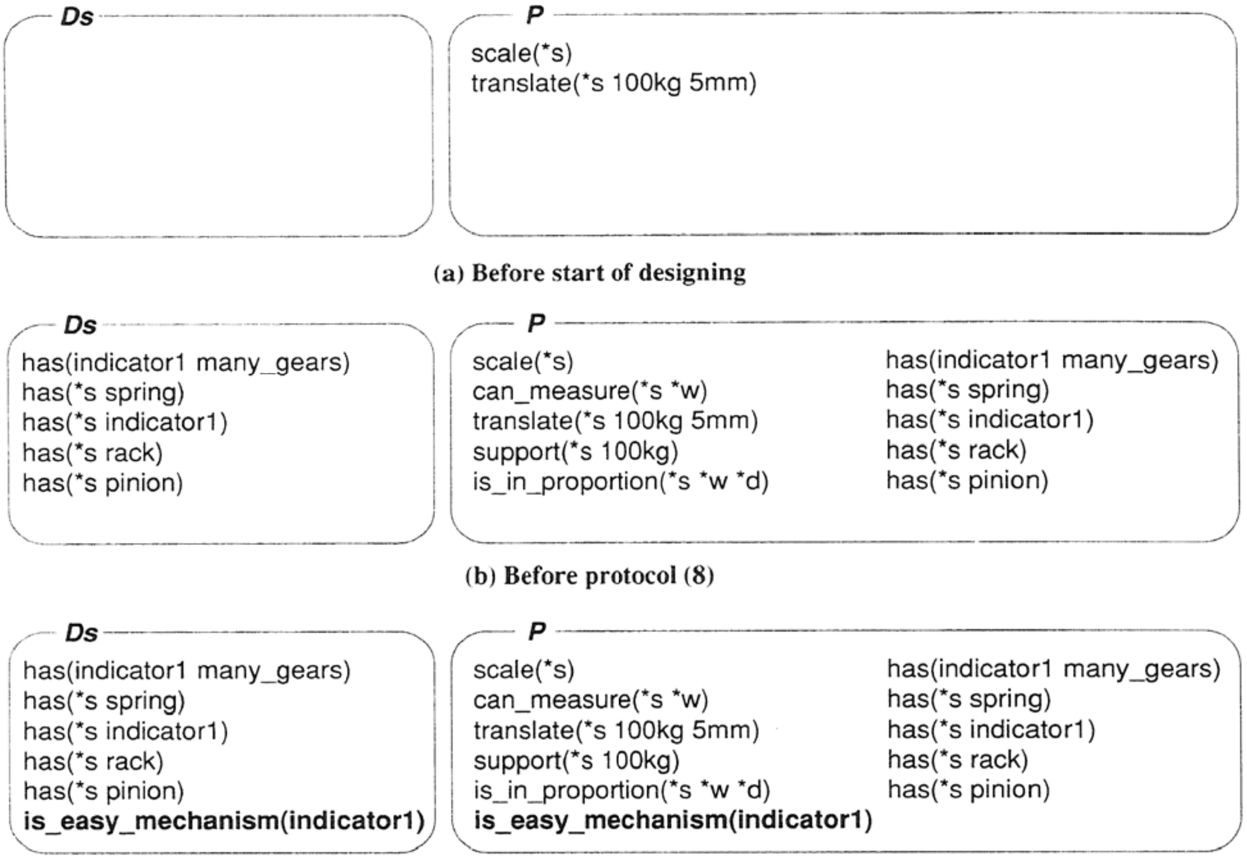
Fig. 15. The contents of *Ko*

Ko'

(2a)' scale(*s) ← can_measure(*s *w) ∧ support(*x *w) ∧ **is_easy_mechanism(indicator1)**
(8a) not(scale(*s)) ← has(*s indicator1) ∧ not(is_easy_mechanism(indicator1)) ∧ **true**

Fig. 16. Modified formulae

This example demonstrates that the designer simulation is capable of replaying design sessions by tracking protocol data obtained by design experiments. This indicates that the computable model presented in this chapter shows the same operational aspects as the cognitive model. It is also justified crucial to employ different types of reasoning for implementing design subprocesses of the cognitive model as a computable model.



```
Ds

P
scale(*s)
translate(*s 100kg 5mm)
```

(a) Before start of designing

```
Ds
has(indicator1 many_gears)
has(*s spring)
has(*s indicator1)
has(*s rack)
has(*s pinion)

P
scale(*s)                         has(indicator1 many_gears)
can_measure(*s *w)                has(*s spring)
translate(*s 100kg 5mm)           has(*s indicator1)
support(*s 100kg)                 has(*s rack)
is_in_proportion(*s *w *d)        has(*s pinion)
```

(b) Before protocol (8)

```
Ds
has(indicator1 many_gears)
has(*s spring)
has(*s indicator1)
has(*s rack)
has(*s pinion)
is_easy_mechanism(indicator1)

P
scale(*s)                         has(indicator1 many_gears)
can_measure(*s *w)                has(*s spring)
translate(*s 100kg 5mm)           has(*s indicator1)
support(*s 100kg)                 has(*s rack)
is_in_proportion(*s *w *d)        has(*s pinion)
is_easy_mechanism(indicator1)
```

(c) After protocol (8)

Fig. 17. Transition of the contents of *Ds* and *P*

7. Discussion

In this article we presented three different models, viz. a descriptive model, a cognitive model, and a computable model. In the descriptive design process model we illustrated a general framework of the design process as an evolutionary refinement process of the metamodel. Since we put emphasis on the representation of the design object and the design process rather than on reasoning aspects in the discussion of the descriptive model, the mechanism of reasoning in design did not become clear. These aspects are highlighted for the discussion on the cognitive design process model. We examined the design processes of the designers, and presented how the designer proceeds her/his design.

Reasoning in design processes is shown as repetition of the design cycle on two levels, i.e. the object level and the action level. Repetition of the design cycle corresponds to the stepwise refinement process presented in the descriptive model. A unit design cycle corresponds to a design step in the descriptive model. Furthermore, the design cycle clarifies different kinds of reasoning that should be done in design processes. The computable design process model realizes these two models by logical reasoning. The framework of the computable model is the evolutionary process of the design object discussed in the descriptive model, i.e, we realized it as a revision process of the theory in logic. This revision process of the theory is performed by logical reasoning, i.e. abduction, deduction, and circumscription, and it is realization of reasoning discussed in the cognitive model. Thus we succeeded in integrating the descriptive and cognitive models using a logical framework, and the result of the integration is shown by the computable model.

Furthermore, based on this computable model we implemented a system called the design simulator. By comparing the sessions performed by the design simulator and those performed by the designer, we obtained information such as what knowledge should be prepared or what kind of reasoning is missing. This information is indispensable to develop the computable model and it also contributes to improve the architecture of intelligent CAD systems.

8. Conclusions

We presented a logical formalization of design processes to establish a computable design process model. The formalization uses both deductive and abductive reasoning, and design is viewed as a process where both of the theory and the goal are gradually enhanced. Furthermore, we introduced circumscription through which a contradictory situation of the current design candidate is removed by modifying the theory. This model is proven computable by the design simulator and it also turned out appropriate for a design process model.

We discussed the relationships among the descriptive, the cognitive, and the computable models. The computable model uses the framework of the descriptive model, while the reasoning in the computable model is an interpretation of the cognitive model. Thus, the model we proposed as a computable model is a realization of both the cognitive model and the descriptive model of design processes.

We showed various approaches to model the design process from a pure theoretical one to a computable one. These approaches are related with each other and they provide an appropriate framework to construct an intelligent CAD system whose behavior is in correspondence with the designer's.

Acknowledgements

We would like to thank Paul ten Hagen for various discussions on design process models, Kazuto Hayashi and Jan Treur for useful discussions on logical formalization, and Juro Baba for contributing to the implementation of the design simulator.

References

1. V. Akman, P.J.W. ten Hagen, J. Rogier, and P.J. Veerkamp, ''Knowledge Engineering in Design,'' *Knowledge-Based Systems*, vol. 1, no. 2, pp. 67-77, 1988.
2. P.T. Cox and T. Pietrzykowski, ''Causes for Events: Their Computation and Applications,'' in *Lecture Notes in Computer Science 230*, pp. 608-621, Springer-Verlag, 1986.
3. R. Coyne, *Logic Models of Design,* Pitman Publishing, London, 1988.
4. T.H. Dietterich and D.G. Ullman, ''FORLOG: A Logic-based Architecture for Design,'' Rep. No. 86-30-8, Computer Science Department, Oregon State University, 1987.

5. K.T. Fann, *Peirce's Theory of Abduction*, Martinus Nijhoff, The Hague, The Netherlands, 1970.

6. N.S. Fann, T.G. Dietterich, and D.R. Corpron, ''Forward Chaining Logic Programming with the ATMS,'' in *Sixth National Conference on Artificial Intelligence*, pp. 24-29, American Association for Artificial Intelligence, 1987.

7. J.J. Finger and M.R. Genesereth, ''RESIDUE: A Deductive Approach to Design Synthesis,'' Technical Report Stan-CS-85-1035, Stanford University, 1985.

8. S. Finger and J.R. Dixon, ''A Review of Research in Mechanical Engineering Design. Part I: Descriptive, Prescriptive, and Computer-Based Models of Design Processes,'' *Research in Engineering Design*, vol. 1, no. 1, pp. 51-67, 1989.

9. M.J. French, *Conceptual Design for Engineers*, Springer-Verlag, Berlin, 1985.

10. V. Goel and P. Pirolli, ''Motivating the Notion of Generic Design within Information-processing Theory: The Design Problem Space,'' *AI magazine*, vol. 10, no. 1, pp. 18-47, 1989.

11. V. Hubka, *Principles of Engineering Design*, Springer-Verlag, Berlin, 1987.

12. V. Lifschitz, ''Computing Circumscription,'' in *Proceedings of Ninth International Joint Conference on Artificial Intelligence, Los Angles, CA*, pp. 121-127, 1985.

13. J.W. Lloyd, *Foundations of Logic Programming*, Springer-Verlag, Berlin, 1984.

14. J. McCarthy, ''Circumscription — A Form of Non-Monotonic Reasoning,'' *Artificial Intelligence*, vol. 13, pp. 27-39, 1980.

15. H. Nakagawa and T. Mori, ''Computable Circumscription in Logic Programming,'' *Transactions of Information Processing Society of Japan*, vol. 28, no. 4, pp. 330-338, 1987. In Japanese

16. D. Poole, ''A Logical Framework for Default Reasoning,'' *Artificial Intelligence*, vol. 36, pp. 27-47, 1988.

17. H. Takeda, T. Tomiyama, and H. Yoshikawa, ''A Logical Formalization of Design Processes,'' in *Intelligent CAD, II*, ed. H. Yoshikawa, and D. Gossard, North-Holland, Amsterdam, 1990.

18. H. Takeda, S. Hamada, T. Tomiyama, and H. Yoshikawa, ''A Cognitive Approach of the Analysis of Design Processes,'' in *The Second ASME Design Theory and Methodology Conference*, 1990. Forthcoming.

19. P.J.W. ten Hagen and T. Tomiyama, *Intelligent CAD systems 1: Theoretical and Methodological Aspects*, Springer-Verlag, Berlin, 1987.

20. T. Tomiyama and H. Yoshikawa, ''Extended General Design Theory,'' in *Design Theory for CAD, Proceedings of the IFIP Working Group 5.2 Working Conference 1985 (Tokyo)*, ed. H. Yoshikawa, and E.A. Warman, pp. 95-130, North-Holland, Amsterdam, 1987.

21. T. Tomiyama, D. Xue, and Y. Ishida, ''An Experience with Developing a Design Knowledge Representation Language,'' in *Intelligent CAD Systems III – Practical Experience and Evaluation*, ed. P.J. Veerkamp, and P.J.W. ten Hagen, Springer-Verlag, Berlin, 1990. Forthcoming.

22. J. Treur, ''A Logical Framework for Design Processes,'' in *Intelligent CAD Systems III – Practical Experience and Evaluation*, ed. P.J. Veerkamp, and P.J.W. ten Hagen, Springer-Verlag, Berlin, 1990. Forthcoming.

23. D.G. Ullman, T.G. Dietterich, and L.A. Stauffer, ''A Model of the Mechanical Design Process Based on Empirical Data: A Summary,'' in *Artificial Intelligence in Engineering Design*, ed. J.S. Gero, pp. 193-215, Elsevier, Amsterdam, 1988.

24. P.J. Veerkamp, R.S.S. Pieters Kwiers, and P.J.W. ten Hagen, ''Design Process Representation in ADDL,'' in *Intelligent CAD Systems III – Practical Experience and Evaluation*, ed. P.J. Veerkamp, and P.J.W. ten Hagen, Springer-Verlag, Berlin, 1990. Forthcoming.

25. P.J. Veerkamp, T. Kiriyama, D. Xue, and T. Tomiyama, ''Representation and Implementation of Design Knowledge for Intelligent CAD – Theoretical Aspects,'' *Proceedings of the Fourth Eurographics Workshop on Intelligent CAD – The Added Value of Intelligence*, Compiegne, 1990.

26. B. Veth, ''An Integrated Data Description Language for Coding Design Knowledge,'' in *Intelligent CAD Systems I – Theoretical and Methodological Aspects*, ed. P.J.W. ten Hagen, and T. Tomiyama, pp. 295-313, Springer-Verlag, Berlin, 1988.

27. R.W. Weyhrauch, ''Prolegomena to a Theory of Mechanized Formal Reasoning,'' *Artificial Intelligence*, vol. 13, pp. 133-170, 1980.

28. D. Xue, T. Kiriyama, P.J. Veerkamp, and T. Tomiyama, ''Representation and Implementation of Design Knowledge for Intelligent CAD – Implementational Aspects,'' *Proceedings of the Fourth Eurographics Workshop on Intelligent CAD – The Added Value of Intelligence*, Compiegne, 1990.

29. H. Yoshikawa, E. Arai, and T. Goto, ''Experimental Design Theory,'' *Journal of the Japan Society of Pecision Engineering*, vol. 47, no. 7, pp. 830-835, 1981. In Japanese

30. H. Yoshikawa, ''General Design Theory and a CAD System,'' in *Man-Machine Communication in CAD/CAM, Proceedings of the IFIP Working Group 5.2 Working Conference 1980 (Tokyo)*, ed. T. Sata, and E.A. Warman, pp. 35-58, North-Holland, Amsterdam, 1981.

31. H. Yoshikawa, ''CAD Framework Guided by General Design Theory,'' in *CAD Systems Framework, Proceedings of IFIP Working Group 5.2*, ed. K. Bo, and E.M. Lillehagen, pp. 241-253, North-Holland, Amsterdam, 1983.