Intelligent Computer Aided Design
D.C. Brown, M. Waldron and H. Yoshikawa (Editors)
Elsevier Science Publishers B.V. (North-Holland)
© 1992 IFIP. All rights reserved.

# An Intelligent Integrated Interactive CAD — A Preliminary Report

Deyi Xue, Hideaki Takeda, Takashi Kiriyama,
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*

This paper describes a preliminary report about a project to develop a third generation intelligent CAD system, called IIICAD (Intelligent Integrated Interactive CAD) system, which is currently conducted at the University of Tokyo. First, we show that from design experiments we can obtain a cognitive design process model that can be transformed to a computable model. Second, we discuss that qualitative physics can be used to describe knowledge about the physical world and to construct a new framework called metamodel for representing design objects. Third, we describe the construction of the IIICAD system that can demonstrate the usefulness of formalizing design processes and of having an integrated data description framework. A prototype of IIICAD is also shown.

## 1. INTRODUCTION

One of the major problems in developing so-called intelligent CAD (Computer Aided Design) systems is the complexity of design knowledge. The only way to overcome this problem is to construct a sound, tough theoretical basis for CAD (Veth, 1987) that consists of theories about design processes, design objects, and design knowledge. Without having these theories, we might be trapped in a pitfall that system development is carried out in an *ad hoc* manner.

The first generation intelligent CAD was based on the expert systems technology (Gero, 1987). Typically, a routine design problem is described in the following way (Brown and Chandrasekaran, 1985). The specifications can be decomposed into elementary subproblems each of which has a solution corresponding to a structural element. A design solution is found by combining these subelements. Thus, the whole design process can be regarded as a problem solving process, and well-known artificial intelligence techniques for problem solving can be used. The second generation intelligent CAD further incorporated abilities for constraint management and solving (Suzuki, Ando, and Kimura, 1990). This means that the design solution as a combination of subelements is now associated with a network of constraints over attributes, properties, etc. For instance, geometric entities cannot exist without fulfilling geometric constraints. Techniques for geometric reasoning and geometrical constraint management became, therefore, an essential part of the second generation intelligent CAD. However, with only these abilities, the system cannot handle more sophisticated information. For example, it is often advocated that constraints represent the designer's intentions that are needed in various design stages (Inui and Kimura, 1990). However, if the designer's intention refers to functional desires, geometric constraints do not suffice to deal with them. A new generation intelligent CAD must be built on deep understanding about both design processes and design objects.

This paper presents our approach towards a new generation of intelligent CAD systems called *IIICAD* (Intelligent Integrated Interactive CAD) (Tomiyama and ten Hagen, 1987). The IIICAD project first started at the Centre for Mathematics and Computer Science (CWI) in Amsterdam (Tomiyama and ten Hagen, 1987; Veth, 1987). Currently, it is continued both at the University of Tokyo and at CWI. For further technical details of each topic, readers are invited to refer to other publications cited in the text.

The rest of the paper is organized as follows. Chapter 2 describes a theory of design processes that we have developed (1) to understand design and (2) to serve as a basis for IIICAD (Takeda, Tomiyama, and Yoshikawa, 1990; Takeda, Veerkamp, Tomiyama, and Yoshikawa, 1990). We began studies about design processes with design experiments. By analyzing the protocols derived from design experiments, we obtained a cognitive design process model. Based on logical formalization of the cognitive design process model, we developed a computable design process model.

Chapter 3 discusses a theory for representing design objects in the IIICAD environment. We employ qualitative physics (Bobrow, 1985; Weld and de Kleer, 1990) to represent knowledge about the physical world and to manage design object models (Kiriyama, Yamamoto, Tomiyama, and Yoshikawa, 1989). For this purpose, we use Qualitative Process Theory (Forbus, 1984) based on which a qualitative reasoning system (Kiriyama, Tomiyama, and Yoshikawa, 1990) was developed.

Chapter 4 is a preliminary report about the implementation of IIICAD. First, the architecture of IIICAD is proposed and the representation of both design processes and design objects are then discussed based on the results of Chapters 2 and 3. In order to represent design knowledge, IIICAD uses a language called IDDL (Veth, 1987; Tomiyama, Xue, and Ishida, 1991) in which design process knowledge is represented in scenarios and divided in two levels, *viz.*, action level and object level. The design object is represented by entities and their relationships, and the metamodel mechanism (Kiriyama, Yamamoto, Tomiyama, and Yoshikawa, 1989) handles the object level information. Chapter 5 concludes the paper.

## 2. THEORY OF DESIGN PROCESSES

Compared with design objects, design processes are not well described nor even understood. In this chapter, we first introduce an evolutionary design process model which is based on General Design Theory (Yoshikawa, 1981; Tomiyama and Yoshikawa, 1987). Then, we build a cognitive design process model based on design experiments. This cognitive model is given a logical formalization and transformed into a computable design process model (Takeda, Veerkamp, Tomiyama, and Yoshikawa, 1990). The computable model serves as the basis of the architecture of IIICAD.

### 2.1. Evolutionary Design Process Model

In General Design Theory (Yoshikawa, 1981; Tomiyama and Yoshikawa, 1987), design is regarded as a mapping from the function space in which design specifications are described in terms of functions onto the attribute space in which design solutions are described in terms of attributes. Roughly speaking, a design process starts with a functional specification of the design object and ends up with a manufacturable description. In non-trivial design, we cannot get a design solution from a design specification directly; design is a stepwise, evolutionary transformation process. This evolutionary process can be modeled by the *metamodel*† concept (Fig. 1).

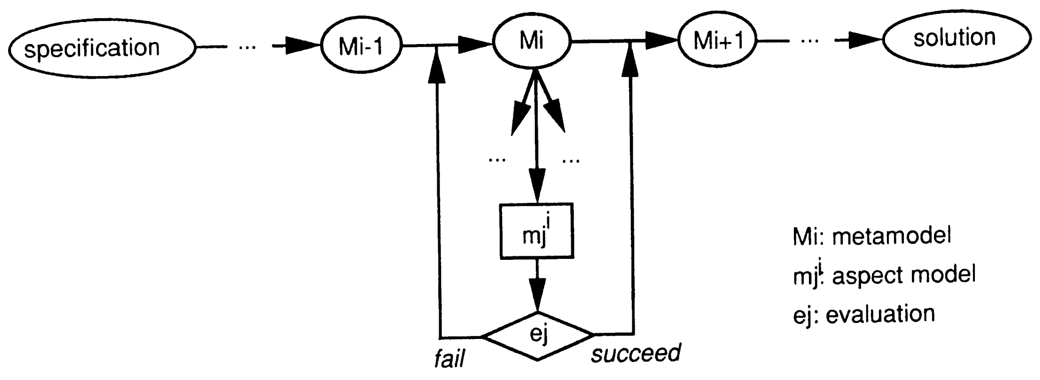


**Figure 1: Evolutionary Design Process Model**

A metamodel reflects the designer's mental model (Johnson-Laird, 1983) about the design object (Fig. 2). When he is given specifications, he roughly imagines how a design solution looks like, what the working principles are, how it is manufactured, etc. He might have preferences, but anyway begins with some working model. He then tries to arrive at a description of a final solution, using sketches, skeletons, symbols, etc. by evaluating ideas in a stepwise manner. This is the evolutionary nature of design. In a practical sense, the metamodel is a database to store all the information about the design object. Aspect models for various kinds of evaluation are derived from this metamodel (see Section 3.2).

† The metamodel concept is used in many different ways throughout this paper. However, they all form a single theory about how to deal with design knowledge.

Every time the metamodel receives a piece of new information about the design object, a new world is created, which represents a design step. Thus, in this evolutionary design process model, design is treated as a sequence of unit design processes. The designer performs these processes by observing the current status of the metamodel and deciding what to do the next, generating candidate solutions, evaluating these candidates, and deciding whether a candidate could be adopted or not. The designer can improve the design object toward the final solution or go back to the previous step.

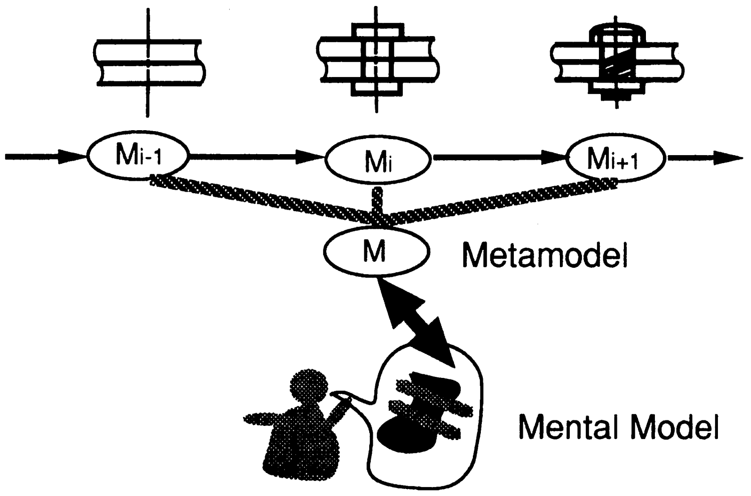


**Figure 2: Metamodel as a Mental Model**

### 2.2. Design Experiment and a Cognitive Design Process Model

In order to understand how design objects evolve, we conducted *design experiments*. This is a kind of psychological experiment in which designers are asked to design a mechanism from a given set of specifications. The whole design session is recorded by a video tape recorder and a CAD-like drawing tool and analyzed with the protocol analysis method. A cognitive design process model is derived from the results of design experiments (Takeda, Tomiyama, and Yoshikawa, 1990). This model shows that a design process is composed of unit design cycles (Fig. 3). Each design cycle has the following five subprocesses:

(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 there is found something unsolved, it becomes a new problem which is solved in another design cycle.

(4) *Evaluation* to evaluate the candidates in various ways, such as structural computation, simulation of behavior, and cost evaluation. If a problem is found as the result of

evaluation, it also 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.

We have also found that design protocols have two different focuses. While there were protocols regarding object level knowledge, we could also observe action level knowledge, e.g., to plan the design process, to decide the knowledge to be used, etc. The entire design process is mainly managed by the action level design process knowledge. This distinction between object level and action level is useful to consider the architecture of IIICAD.

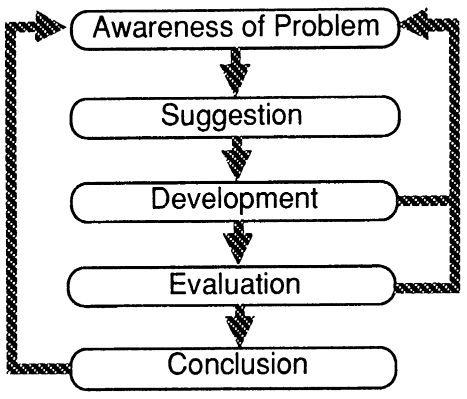


**Figure 3: Design Cycle**

## 2.3. A Logical and Computable Design Process Model

In order to represent design process knowledge in IIICAD, a computable model to describe design cycles is developed (Takeda, Veerkamp, Tomiyama, and Yoshikawa, 1990). Design processes are logically formalized as follows:

$$D_s \cup K_o \vdash P,$$

where $D_s$ is a set of logical formulae describing design solutions that the designer want to obtain, $K_o$ is knowledge on object properties and behaviors, and $P$ is properties of design solutions, respectively. Required specifications are included in $P$. First assuming that $K_o$ is the design knowledge and $P$ is the specifications, the designer tries to obtain a candidate solution with abduction (Fann, 1970). Then by using deduction, the candidate is detailed. If the candidate does not satisfy the specifications, the designer either tries alternative candidates, or modifies design knowledge or specifications.

The cognitive design process model can be interpreted in the context of this computable model. The suggestion subprocess is an abductive process to obtain $D_s$ from $P$ and $K_o$. The development and evaluation subprocesses are deductive processes to obtain $P$ from $D_s$ and $K_o$. In the development and evaluation subprocesses, because of the incompleteness of knowledge base, contradictions sometimes occur. These contradictions are

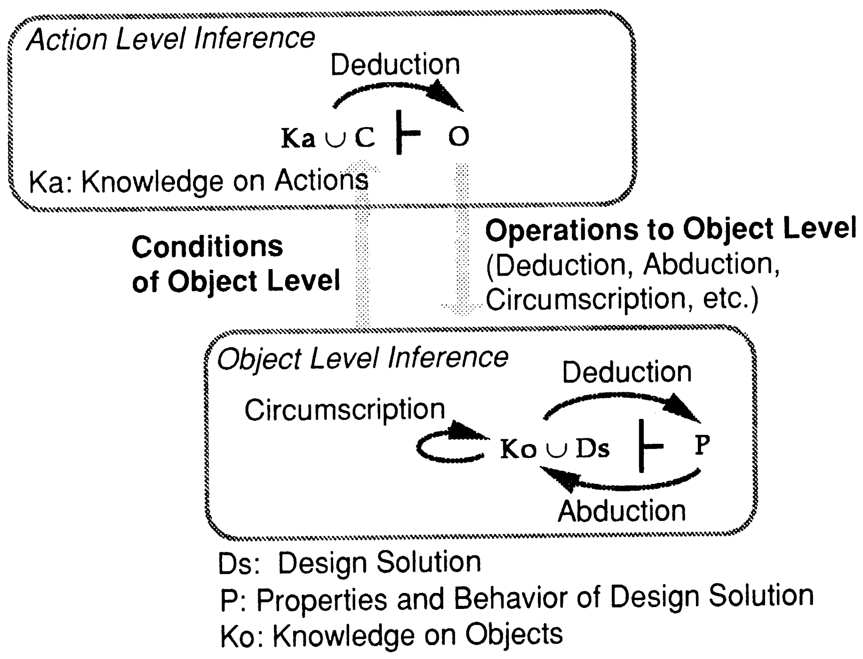


**Figure 4: Two Levels of Design Process Knowledge**

considered exceptions and handled by circumscription (McCarthy, 1980), so that the knowledge on object can be modified to subsume the exceptions. If the required properties cannot be derived because of the modifications of the knowledge base, it should be defined as a new problem. This is a jump to an a wareness-of-problem subprocess.

As mentioned in the previous section, we found two different levels in the design protocols (Fig. 4). In the computable design process model, the action level reasoning system selects the most appropriate knowledge base and schedules the most appropriate object level reasoning method, based on reports from the object level reasoning system. This meta-level inference structure drives the evolution of the design object.

## 3. THEORY OF DESIGN OBJECTS

Although the representation of design objects is perhaps the most developed area in CAD studies, there are still problems to be solved, such as

- to describe knowledge about the physical world that is crucial for creative design, and
- to describe a design object in an integrated way.

In this chapter, we first introduce qualitative physics to symbolically describe and reason about dynamic physical phenomena (Bobrow, 1985; Weld and de Kleer, 1990). Then we introduce a mechanism called *metamodel* mechanism as a fundamental framework for modeling design objects (Kiriyama, Yamamoto, Tomiyama, and Yoshikawa, 1989). Here, once again we use the concept of *metamodel* that was once introduced in the evolutionary design process model. Techniques of qualitative physics allow for symbolic descriptions that reflect the designer's mental model and the physical restrictions imposed on the design object. Using knowledge about the physical world, the metamodel mechanism maintains integration and consistency of various aspect models and supports the designer in an intelligent way.

## 3.1. Qualitative Process Theory

Qualitative physics (Bobrow, 1985; Weld and de Kleer, 1990) serves as a framework to symbolically describe knowledge about the physical world. It has two roles, *viz.*, *modeling* structure of a physical system and *reasoning* about its dynamic behaviors. We employ the basic ideas of *Qualitative Process Theory* (Forbus, 1984) to represent physical phenomena. The theory consists of the following notions. An *individual* represents an entity existing in the physical world. An *individual view* denotes how a set of individuals are seen from a particular point of view. For example, a heater is an individual view that refers to any entity which can be used as a heat source. A *process* is a description about a physical phenomenon that influences on the system. We have developed a qualitative reasoning system based on Qualitative Process Theory on Smalltalk-80 (Kiriyama, Tomiyama, and Yoshikawa, 1990) and this system plays a central role in the metamodel mechanism. Our qualitative reasoning system uses de Kleer's ATMS (Assumption-based Truth Maintenance System) (de Kleer, 1986) to maintain information about different feasible situations. Due to this, the system is capable of finding out identical situations; for instance, it can conclude that a motor rotates, instead of generating endless identical situations.

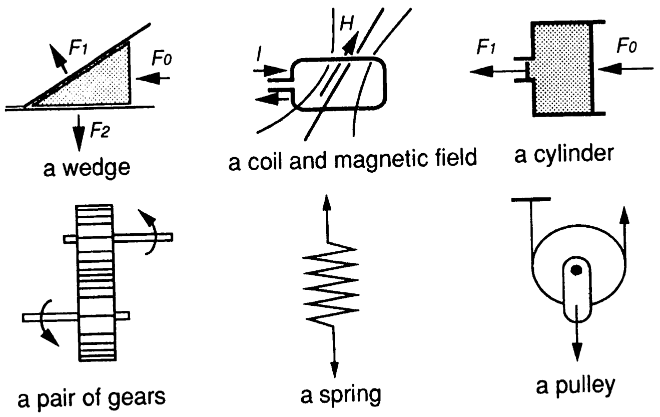
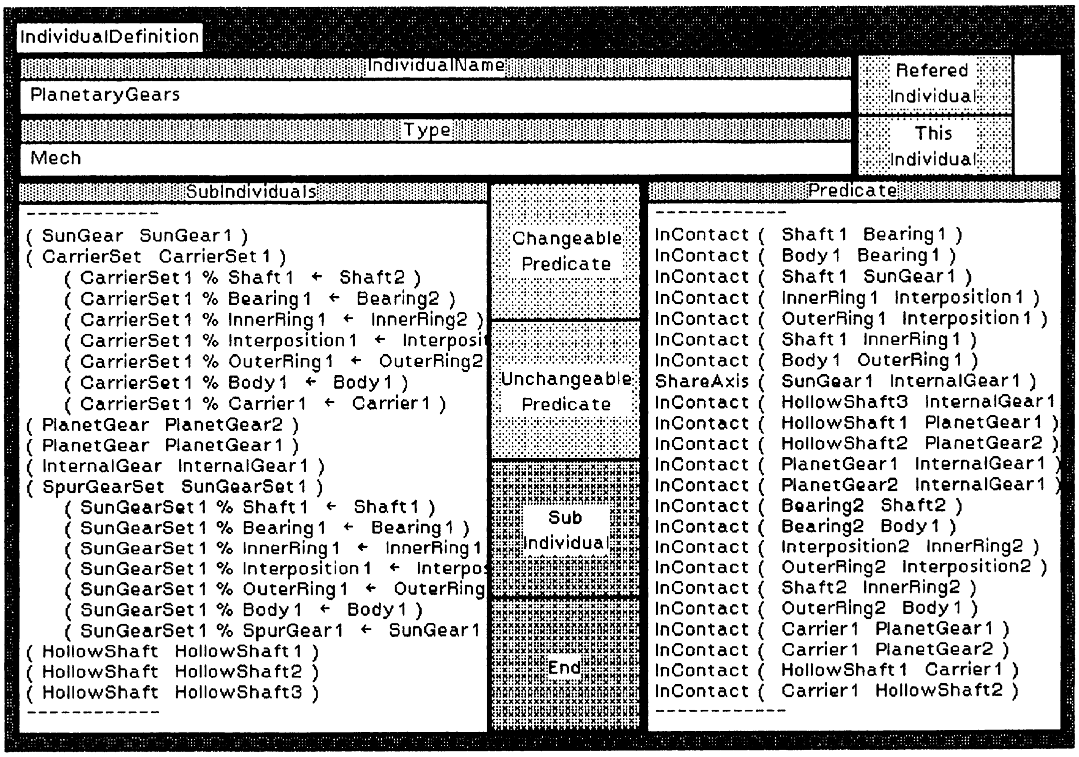


**Figure 5: Physical Features**

Here, we introduce the concept of *physical features* (Kiriyama, Yamamoto, Tomiyama, and Yoshikawa, 1989). The concept of *features* (Dixon and Cunningham, 1989) is now considered vital for representing teleological knowledge about the relationship between a concept and attributive information. Similarly, a physical feature describes knowledge about a physical phenomenon and attributes (Fig. 5). For example, a wedge is a physical feature that describes the relationship between two intersecting surfaces and physical phenomena, such as force diversion and friction angle. We use Qualitative Process Theory to represent physical features. The components of a physical feature are represented by individual views and physical phenomena by process views.



```
PlanetaryGears
```

```
Mech
```

```
------------
( SunGear  SunGear1 )
( CarrierSet  CarrierSet1 )
    ( CarrierSet1 % Shaft1  ←  Shaft2 )
    ( CarrierSet1 % Bearing1  ←  Bearing2 )
    ( CarrierSet1 % InnerRing1  ←  InnerRing2 )
    ( CarrierSet1 % Interposition1  ←  Interposi
    ( CarrierSet1 % OuterRing1  ←  OuterRing2
    ( CarrierSet1 % Body1  ←  Body1 )
    ( CarrierSet1 % Carrier1  ←  Carrier1 )
( PlanetGear  PlanetGear2 )
( PlanetGear  PlanetGear1 )
( InternalGear  InternalGear1 )
( SpurGearSet  SunGearSet1 )
    ( SunGearSet1 % Shaft1  ←  Shaft1 )
    ( SunGearSet1 % Bearing1  ←  Bearing1 )
    ( SunGearSet1 % InnerRing1  ←  InnerRing1
    ( SunGearSet1 % Interposition1  ←  Interpo
    ( SunGearSet1 % OuterRing1  ←  OuterRing
    ( SunGearSet1 % Body1  ←  Body1 )
    ( SunGearSet1 % SpurGear1  ←  SunGear1
( HollowShaft  HollowShaft1 )
( HollowShaft  HollowShaft2 )
( HollowShaft  HollowShaft3 )
------------
```

```
------------
InContact (  Shaft1  Bearing1 )
InContact (  Body1  Bearing1 )
InContact (  Shaft1  SunGear1 )
InContact (  InnerRing1  Interposition1 )
InContact (  OuterRing1  Interposition1 )
InContact (  Shaft1  InnerRing1 )
InContact (  Body1  OuterRing1 )
ShareAxis (  SunGear1  InternalGear1 )
InContact (  HollowShaft3  InternalGear1
InContact (  HollowShaft1  PlanetGear1 )
InContact (  HollowShaft2  PlanetGear2 )
InContact (  PlanetGear1  InternalGear1 )
InContact (  PlanetGear2  InternalGear1 )
InContact (  Bearing2  Shaft2 )
InContact (  Bearing2  Body1 )
InContact (  Interposition2  InnerRing2 )
InContact (  OuterRing2  Interposition2 )
InContact (  Shaft2  InnerRing2 )
InContact (  OuterRing2  Body1 )
InContact (  Carrier1  PlanetGear1 )
InContact (  Carrier1  PlanetGear2 )
InContact (  HollowShaft1  Carrier1 )
InContact (  Carrier1  HollowShaft2 )
------------
```

**Figure 6: Physical Feature Editor**

We are currently conducting a project to build a physical feature knowledge base that may contain of the order of 10,000 knowledge chunks. This knowledge base, together with the qualitative reasoning system mentioned above, is used in IIICAD to model a design object and to reason about its behaviors. Figure 6 shows an example physical feature description as appears on the physical feature editor. At the time of writing this paper, we have in the database about two thousand chunks of knowledge about physical rules and basic kinematics that are considered common sense for engineering designers.

Lenat *et al.* at MCC are conducting the Cyc project aiming at building a large scale common sense knowledge base (Lenat and Guha, 1989). Feigenbaum *et al.* at Stanford University have started the *How Things Work* project to collect knowledge about physical devices for engineering design, diagnosis, etc. (Cutkosky and Tenenbaum, 1990). Our project is similar to these projects but differs in that we have our own inference engine (the Cyc project aims at much bigger goal) and that we use Qualitative Process Theory as the *ontology* (Hayes, 1985). (The Cyc and How Things Work projects have different ontology.)

## 3.2. Metamodel Mechanism to Integrate Design Object Models

During the course of design, the design object has to be described and evaluated from various points of view. These models, called *aspect models,* include a geometric model, a kinematic model, a dynamic model, etc., and are not independent from each other. We consider that we observe a physical phenomenon associated with the design object and create an *aspect model* focusing on only interesting particular properties and attributes. All other properties and attributes are ignored. This modeling process takes place according to a particular *background theory* and an aspect model can have different representations for different purposes (Fig. 7). For example, a geometric model is composed of geometric elements such as vertices, edges, faces and solid objects. Its background theory is algebraic geometry.

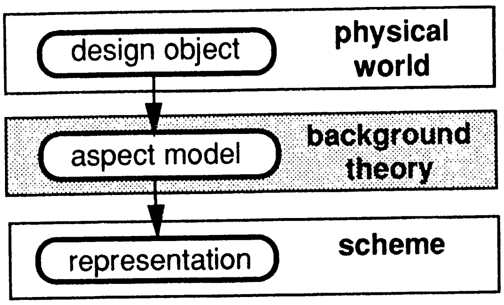


**Figure 7: Physical World, Model, and Representation**

The metamodel mechanism has the following three roles, and by doing so reflects the designer's mental model about the design object. First, it serves as a central model of the design object by maintaining relationships among various kinds of aspect models. Second, it provides a qualitative model of the design object as a combination of physical features. Third, it provides a design workspace for the evolutionary design process.

The metamodel mechanism contains a model builder, an aspect model generator, and a consistency manager. The model builder constructs a network of concepts relevant to the design object from a primary model that is a combination of physical features (Fig. 8). This primary model can be constructed from functional information by the user using the computable design process model. We define a function as an abstract description of behavior and are developing a function modeler that can handle the trichotomy of function, behavior, and structure based on the qualitative reasoning system (Umeda, Takeda, Tomiyama, and Yoshikawa, 1990). The qualitative reasoning system reasons about all possible behaviors of the design object from the primary model and constructs a network of concepts (Fig. 9). This network contains concepts about behaviors, physical features, relationships among these concepts. Thus, the concept network is reflection of the mental model of the designer.

The aspect model generator generates an aspect model of the design object by selecting only relevant concepts to the aspect model from the concept network. In this context, an aspect model is a partial description of the concept network, such that it contains only information about the object from a particular point of view. The information of an aspect model can be further converted to be used by other modelers (such as geometric modelers and numerical computation systems) for evaluation. This means that the metamodel mechanism provides a pluggable environment that is different from product modeling.

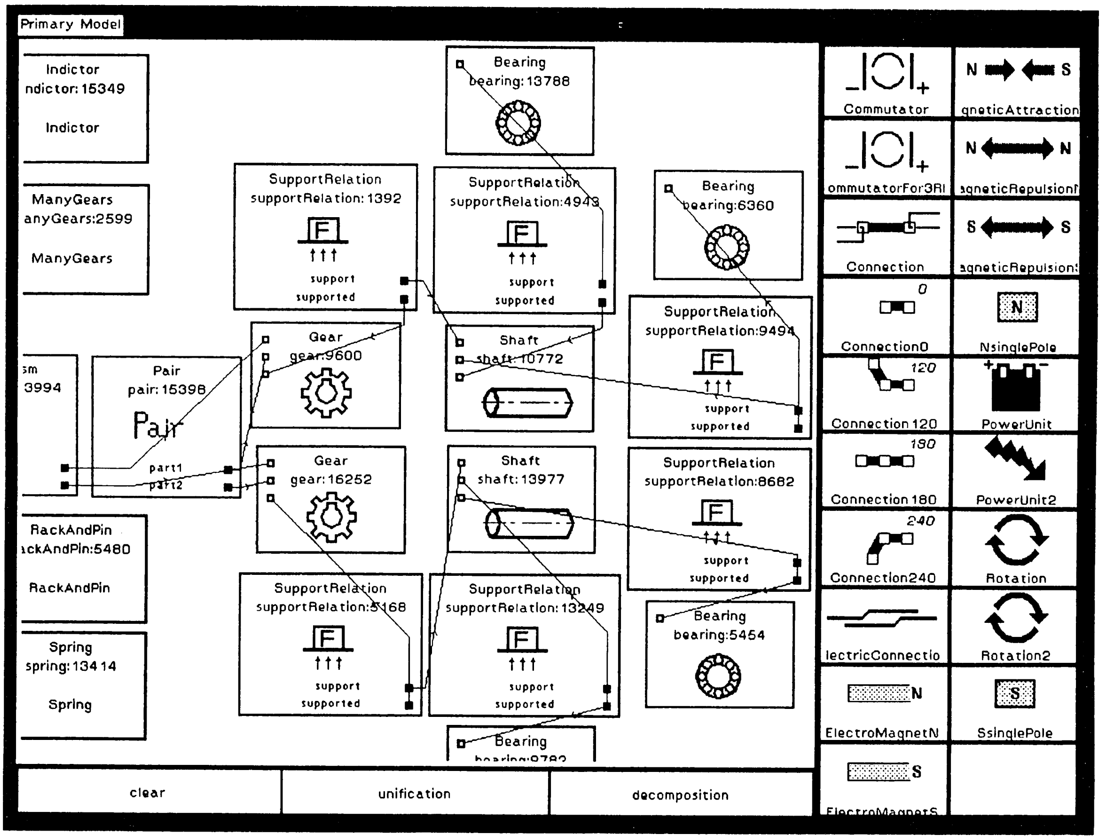


**Figure 8: Primary Model**

The consistency manager maintains consistency among aspect models and propagates modifications.

## 4. IMPLEMENTING INTELLIGENT INTEGRATED INTERACTIVE CAD

In the previous chapters, we briefly outlined theories and techniques for representing knowledge about design processes and design objects in IIICAD. Integrating these results, this chapter discusses the architecture of IIICAD and illustrates how design processes and design objects are represented. Finally, we show the implementation method for this system and give an example about how IIICAD can be used.

### 4.1. System Architecture

Figure 10 depicts the architecture of IIICAD. It has three major components, *viz.*, a *design process handling system*, a *metamodel mechanism*, and a *user interface manager*.

Design process knowledge is described in *scenarios*. A *scenario*, written in *IDDL* (Integrated Data Description Language) (Veth, 1987; Tomiyama, Xue, and Ishida, 1991) is a set of rules and divided into two levels, *viz.*, action level and object level. Action level scenarios and object level scenarios are programmed in *action level knowledge base browser* and *object level knowledge base browser*, respectively. The *supervisor* has two

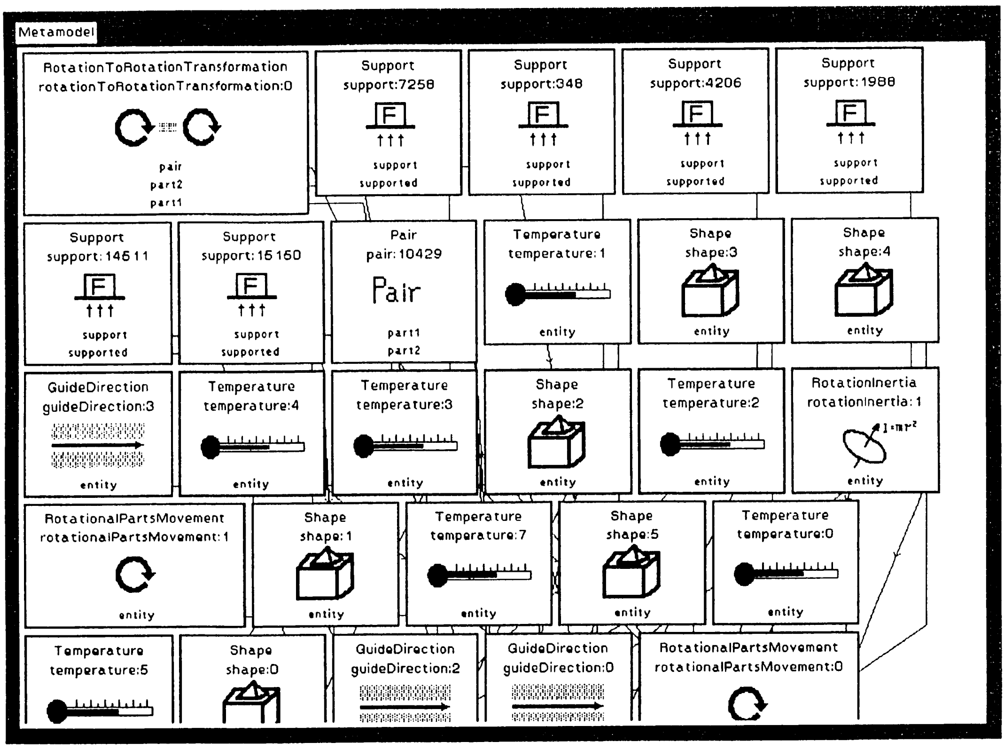


**Figure 9: Concept Network**

design process managers, i.e., an *action level manager* and an *object level manager.* The action level manager receives reports from the object level manager and controls the object level design process by executing action level scenarios from the action level scenario base, i.e., selecting the most appropriate object level scenarios from the object level scenario base, choosing the most appropriate reasoning method, and permitting interactions with the designer through the *design browser.* In fact it proceeds the action level design process step by step, asking for the designer's confirmation.

The action level design process knowledge describes relationships between the situations of the metamodel mechanism and the suitable operations to be performed. This means that the action level manager performs the design cycle, i.e., awareness-of-problem, suggestion, development, evaluation, and conclusion subprocesses introduced in Section 2.2.

The object level manager solves object level problems by using knowledge about particular types of object that is described in selected scenarios. There are three inference system in the object level manager, i.e., an *abduction system,* a *deduction system* and a *circumscription system* to perform different kinds of operations.

Operations to the design object are realized by the metamodel mechanism. As discussed in Chapter 3.2, it has a *model builder,* an *aspect model generator,* and a *consistency manager.* The supervisor requests the model builder to add new information about the design object that can be obtained either from the design browser, from aspect models through the consistency manager, or by constructing a concept network about the design

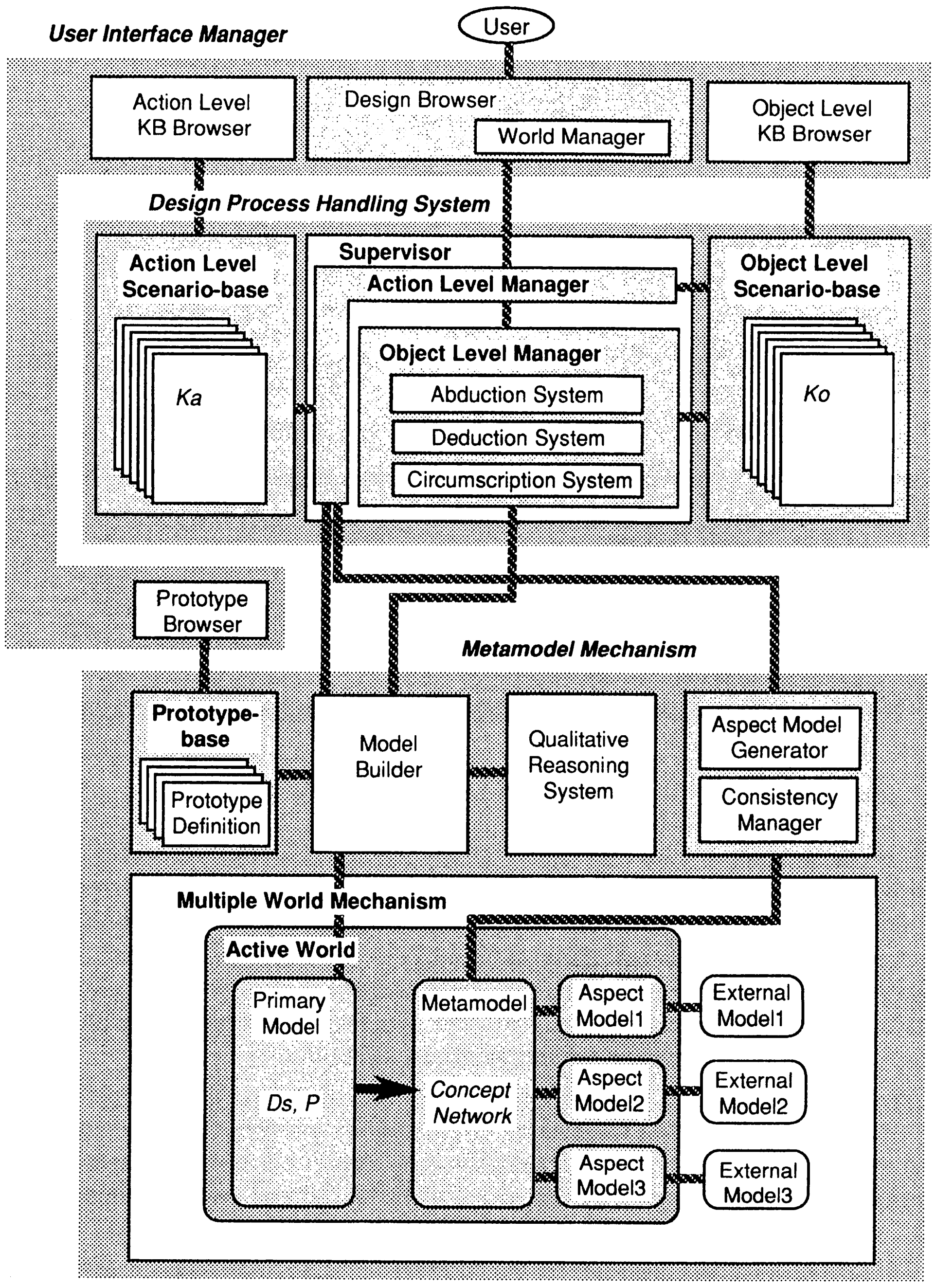


**Figure 10: System Architecture**

object from the primary model. The aspect model generator is used to generate aspect models based on descriptions about external modelers (such as a geometric modeler). The consistency manager maintains integration among different aspect models. If there is a modification to the design object, it is reported to the supervisor and, if necessary, the consistency manager tries to keep the consistency through the aspect model generator. A standard description about an entity or a physical phenomenon is called a *prototype* which is programmed in the *prototype browser.* Prototypes can be used to build a primary model, or to construct a metamodel from the primary model.

A design object is described in several worlds (called *multiple world mechanism)* (Veerkamp, 1989) to realize its evolution. Only one world is *active* and operations are performed only in this active world. Worlds are organized into an environment called *world manager.* A world has both a *primary model* and a *metamodel* generated from the primary model. A primary model consists of *objects, facts* and *selected scenarios.* Objects and facts written in IDDL are used to describe the entities and the relationships among them. Selected scenarios are the knowledge selected from the object level scenario base. Objects and facts in a primary model are organized by ATMS. The metamodel is generated from the primary model by using the qualitative reasoning system (see Section 3.2). The data in a metamodel is also organized by ATMS. These two ATMS systems cooperate, such that suggestions in a design cycle are treated as assumptions for the metamodel mechanism.

In IIICAD, an evolutionary design process takes place in the following way (see Fig. 11). Suppose at a certain step in a design process we have metamodel† $M_{i-1}$ that consists of three facts, $a$, $b$, and $c$, and we select $c$ as the problem to be solved. The action level manager of the supervisor selects an appropriate object level scenario from the object level scenario base to make suggestions for this problem. This object level scenario becomes an selected scenario and a suggestion $s1$ is made, which is added to the database by the model builder. Then, suggestion $s1$ is developed by using suitable object level scenarios chosen by the action level manager and an aspect model $m_1{}^i$ is generated by the aspect model generator. If evaluation for $s1$ is a failure, all of the data related to this suggestion should be removed by using ATMS. If evaluation for another suggestion $s2$ is a success, then the metamodel evolves to $M_{i+1}$ by adding $s2$.

## 4.2. Representation of Design Processes

### 4.2.1. Scenario

In IDDL, a scenario is composed of a set of rules and executed in much the same way as conventional production rule systems (for more information see (Tomiyama, Xue, and Ishida, 1991)). A rule has a well-known syntax of

IF <*conditions*> THEN <*results*>.

Both of <*conditions*> and <*results*> are a list of predicates connected with & (logical and) and | (logical or). A predicate may be preceded by a negator ~. We have objects to

† Here *metamodel* denotes the whole database that contains all the information about the design object.

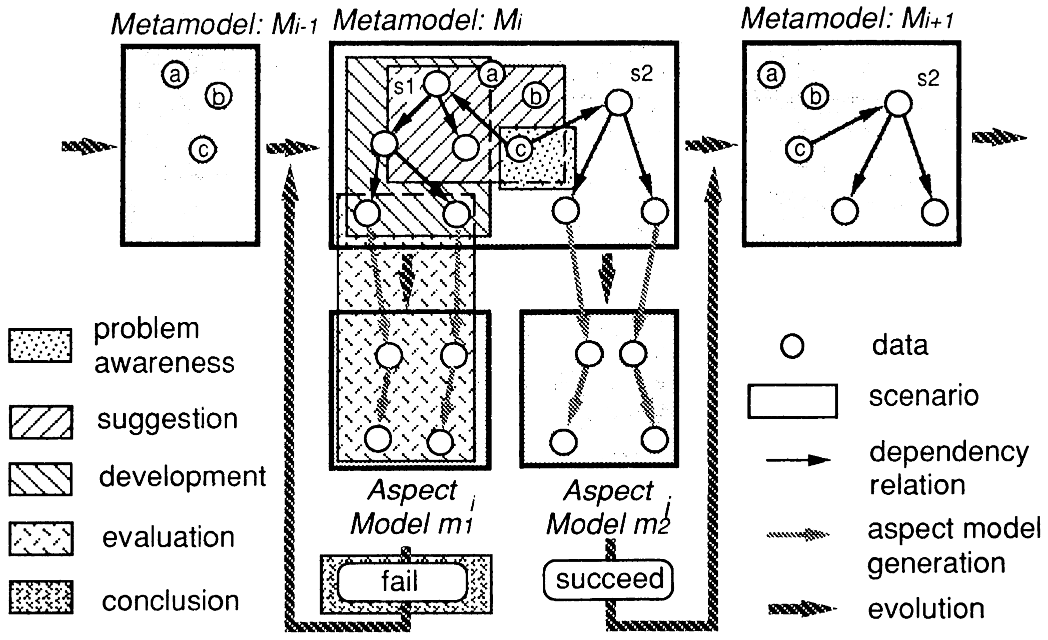


**Figure 11: Design Process with Metamodel Evolution**

denote entities, predicates (or facts) to denote relationships among entities, and functions (or attributes) to denote properties of entities (see Section 4.3).

A rule can be used in *forward reasoning* for deduction and in *backward reasoning* for abduction. (Strictly speaking, abduction should not be realized by simple backward reasoning.) ATMS is available to deal with reasoning concerned with assumptions.

An action level scenario can also execute other action level scenarios by built-in predicate *use*. After the execution of such scenarios, the execution control will be returned to the original scenario. For choosing the most appropriate scenario from the object level scenario base for a given situation, we use a built-in predicate *select*. Built-in predicate *do* is used to perform object level operations such as abduction, deduction or circumscription.

### 4.2.2. Representation of Design Processes Knowledge

In IIICAD, design process knowledge is described both in the action level and in the object level. The action level design process knowledge is used by the action level manager to control the design process and has more general and abstract descriptions than the object level design process knowledge.

Figure 12 shows the top action level scenario that basically executes the design cycle. Every time the action level manager finds matching rules, it asks for the designer's confirmation. This suggests that the designer and the supervisor *design a design process*, which can be realized by introducing a second-order predicate logic capability. The scenario should be read in the following way.

```
topActionLevelScenario
 "Top level operation"

BEGIN
IF TRUE THEN use(addSpecification),
 "(1) Add specification"
IF TRUE THEN use(problemAwareness),
 "(2) Try to set a problem"
IF problem(X) THEN use(suggestion),
 "(3) Try to make suggestion"
IF suggestion(X) THEN use(development),
 "(4) Try to make development"
IF suggestion(X) THEN use(evaluation),
 "(5) Try to make evaluation"
IF suggestion(X) THEN use(conclusion),
 "(6) Try to make conclusion"
IF contradiction(X)
   THEN use(contradictionResolution),
 "(7) Try to solve contradiction"
IF suggestion(X) & metamodel(X)
   THEN do(metaToAspectGeneration),
 "(8) Make aspect model from metamodel"
IF suggestion(X) & metamodel(X)
   THEN do(consistencyInspection),
 "(9) Inspect integration of aspect models"
... ...
END
```

**Figure 12: A Top Action Level Scenario**

*(1) Specifications can be added at any time.*
*(2) Problems can be selected at any time.*
*(3) If there are problems, suggestions should be made.*
*(4) If there are suggestions, developments can be made.*
*(5) If there are suggestions, evaluations can be made.*
*(6) If there are suggestions, conclusions can be made.*
*(7) If there are contradictions, these contradictions should be solved,*
*(8) If there are suggestions in metamodel, aspect models can be generated, etc.*

(1) *Specification Definition.*

Specifications are described in terms of objects, facts, and attributes, and placed in the primary model.

(2) *Suggestion.*

Figure 13 shows an action level scenario for giving suggestions to a problem. First, the action level manager selects the most appropriate object level scenario from the object level scenario base to make suggestions to the problem. If there exist some, these scenarios should be used by the object level manager; if not, the user should make a suggestion. If a suggestion is made, the problem should not be focused again.

```
suggestion
 "Making suggestion to problem"

BEGIN
IF problem(X) & scenario(Y)
   & canSuggestByAbduction(Y,X)
   THEN select(Y)
          & do(suggestionByAbduction)
          & ~problem(X),
 "Find suggestion by abduction"
IF problem(X) & scenario(Y)
   & canSuggestByFunctionDescription(Y,X)
   THEN select(Y)
          & do(suggestionByFunctionDescription)
          & ~problem(X),
 "Find suggestion by function description"
IF problem(X) & canNotBeSuggested(X)
  THEN use(makingNewKnowledge),
 "Make new knowledge"
IF TRUE THEN succeed,
 "END"
END
```

**Figure 13: An Action Level Scenario Used for Suggestion**

Object level scenarios are selected and used in the following manner. Suppose we have a problem *rotationPair*$(r_1, r_2)$ to be solved. In the *suggestion* scenario, several methods are described in order to get suggestions to the selected problem (Fig. 13). For example, the object level scenarios shown in Fig. 14 can be used for making suggestions such as a *beltGearDrive*$(r_1, r_2)$ or a *gearPair*$(r_1, r_2)$ separately by backward reasoning.

```
beltGearDriveDescription
BEGIN
IF beltGearDrive(X,Y)
   THEN rotationPair(X,Y),
IF beltGearDrive(X,Y)
   THEN sameDirection(X,Y),
... ...
END
```

**(a) "beltGearDriveDescription" Scenario**

```
gearPairDescription
BEGIN
IF gearPair(X,Y)
   THEN rotationPair(X,Y),
IF gearPair(X,Y)
   THEN ~sameDirection(X,Y),
... ...
END
```

**(b) "gearPairDescription" Scenario**

**Figure 14: Object Level Scenarios Used for Suggestion**

(3) *Development.*

Detailed descriptions of a suggestion are obtained in the development subprocess. The scenarios in Fig. 14 can also be used for development purpose by forward reasoning such as to conclude that if the suggestion is a $beltGearDrive(r_1, r_2)$, it has the property of $sameDirection(r_1, r_2)$. Here the two rules (see Fig. 14 (a)) represent the knowledge $K_o$, $beltGearDrive(r_1, r_2)$ represent the design solution $D_s$, $rotationPair(r_1, r_2)$ and $sameDirection(r_1, r_2)$ represent the properties of design object $P$ (see Fig. 4). The construction of the metamodel from a primary model is also considered as a development.

(4) *Evaluation.*

The evaluation of a suggestion is performed in the following way. First, the evaluation criteria should be determined. The specifications, such as constraints for attributes, can be used as the evaluation criteria. The evaluation criteria are evaluated one by one until all the criteria are satisfied. If there is not enough information for evaluation, operations, e.g., to obtain more information from further development or to generate an aspect model, are needed. The later is done by the aspect model generator and we can use external application systems (such as a geometric modeling system) for evaluation. After the evaluation, we know whether the suggestion is good or not.

(5) *Conclusion.*

If the result of evaluation is unsatisfactory, unsuitable suggestions and relevant information are removed from the database by using ATMS. If satisfactory, we perform a comparison among these suggestions. If we have no suggestion, the problem should be focused again, i.e., we must go back to the suggestion subprocess or start a new design cycle from the awareness-of-problem subprocess.

(6) *Contradiction Resolution.*

Unsatisfactory results of evaluation can be regarded as contradictions of knowledge. These contradictions can be resolved by predicate circumscription† in which we calculate unifications for the abnormal predicates in the condition part of rules. After circumscription, we should check whether the problem can be concluded again. If the problem cannot be concluded or the contradiction cannot be solved, it is considered that the suggestion is not suitable, and the problem should be focused again. The suggestion should also be removed from the database.

## 4.3. Representation of Design Objects

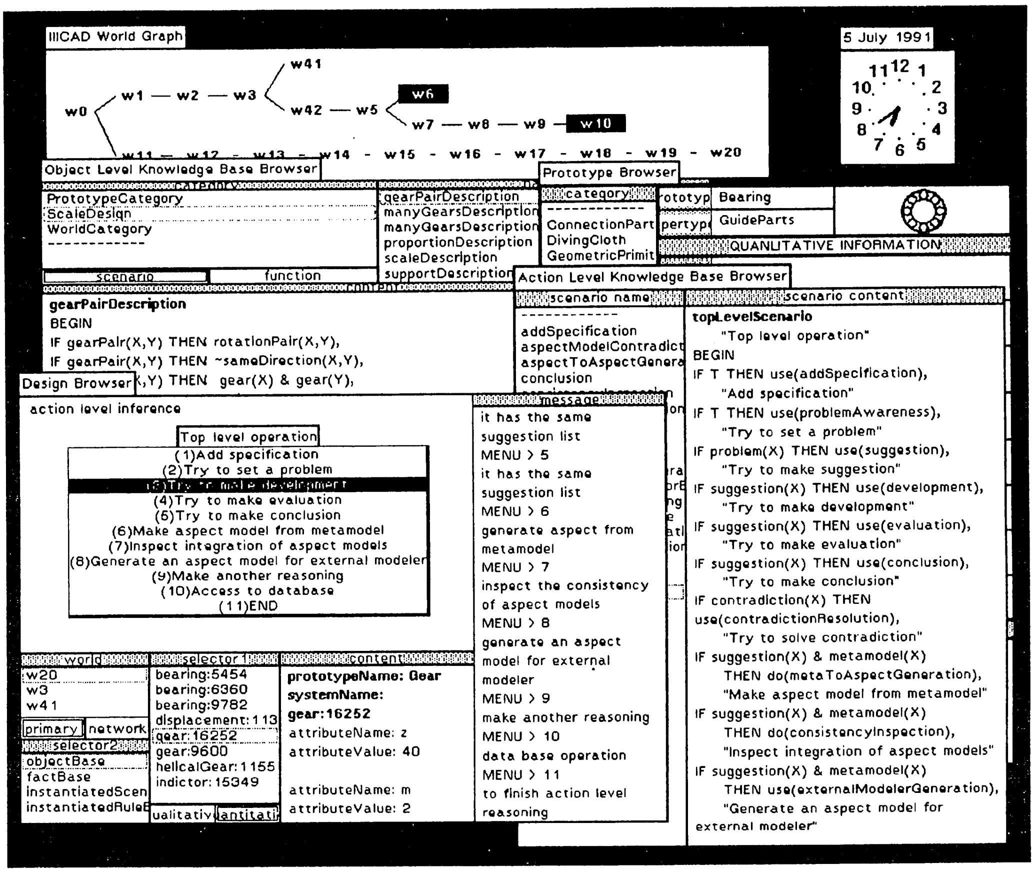
Design objects are described by *objects*, *facts*, and *functions* written in IDDL (Tomiyama, 1989; Tomiyama, Xue, and Ishida, 1991). Objects are instantiated from *prototypes* which have standard descriptions about entities and physical phenomena. There are two kinds of information in a prototype. Qualitative information is described by a *process* which consists of both the premise and the physical phenomenon that influences on the system. Quantitative information is described by *attributes* and *functions* over these attributes.

† We use a circumscription algorithm proposed by (Nakagawa and Mori, 1987). They developed an algorithm for computing circumscription (Lifschitz, 1985). For technical details, refer to (Takeda, Veerkamp, Tomiyama, and Yoshikawa, 1990).

Prototypes are collected as physical features and, as described in Section 3.1., currently we are building a large physical feature knowledge base. The instantiation of an object from a prototype is performed by the model builder. A fact describes a property or a relationship between objects. Facts are described in predicates which are suitable for logical reasoning. Design specifications $P$ and design solutions $D_s$ are described in facts.

## 4.4. IIICAD System Implementation

Currently we are developing a prototype system of IIICAD in Smalltalk-80† (Goldberg and Robson, 1983).



```
gearPairDescription
BEGIN
IF gearPair(X,Y) THEN rotationPair(X,Y),
IF gearPair(X,Y) THEN ~sameDirection(X,Y),
...,Y) THEN  gear(X) & gear(Y),
```

```
topLevelScenario
    "Top level operation"
BEGIN
IF T THEN use(addSpecification),
    "Add specification"
IF T THEN use(problemAwareness),
    "Try to set a problem"
IF problem(X) THEN use(suggestion),
    "Try to make suggestion"
IF suggestion(X) THEN use(development),
    "Try to make development"
IF suggestion(X) THEN use(evaluation),
    "Try to make evaluation"
IF suggestion(X) THEN use(conclusion),
    "Try to make conclusion"
IF contradiction(X) THEN
use(contradictionResolution),
    "Try to solve contradiction"
IF suggestion(X) & metamodel(X)
    THEN do(metaToAspectGeneration),
    "Make aspect model from metamodel"
IF suggestion(X) & metamodel(X)
    THEN do(consistencyInspection),
    "Inspect integration of aspect models"
IF suggestion(X) & metamodel(X)
    THEN use(externalModelerGeneration),
    "Generate an aspect model for
external modeler"
```

```
it has the same
suggestion list
MENU > 5
it has the same
suggestion list
MENU > 6
generate aspect from
metamodel
MENU > 7
inspect the consistency
of aspect models
MENU > 8
generate an aspect
model for external
modeler
MENU > 9
make another reasoning
MENU > 10
data base operation
MENU > 11
to finish action level
reasoning
```

```
prototypeName: Gear
systemName:
gear:16252
attributeName: z
attributeValue: 40

attributeName: m
attributeValue: 2
```

**Figure 15: A Hardcopy of the IIICAD System**

Figure 15 shows a hardcopy of this prototype system. In the IIICAD system, *Design Browser* is the environment to perform a design work. The action level inference system gives the possible operations with a menu by using action level scenarios. Then by performing the operations selected by the user, object level scenarios are selected and used to

† Smalltalk-80 is a Registered Trade Mark of Xerox Corp.

evolve the design object.

## 4.5. An Example

This section illustrates an example about how design is performed in the IIICAD system. The knowledge used in this example comes from protocol data obtained in a design experiment. The task is to design a weighing scale. Figure 16 shows the protocol data about this task and Figure 17 shows the formalized knowledge written in production rules. The rules are described in several scenarios and used when needed. For the convenience of explanation, each rule is labeled with a rule number.

(1) What mechanism does a standard scale use?
(2) It measures the displacement caused by weight like this (Figure A).
(3) So we should use a spring.
(4) We can also use a spring this way (Figure B).
(5) If we use a rack and pinion (Figure C), we can measure the weight, because the displacement is in proportion to the weight.
(6) As it translates 100kg of the weight to 5mm of the displacement, it is impossible to realize it with this mechanism.
(7) If we do not mind the accuracy, it is possible to use many gears .
(8) But we cannot see the indictor upward.
(9) The indictor is difficult to see either.
(10) The indictor in a standard scale is easy to see.
(11) If the indictor can be seen upward, it is easy to see.
(12) How about using a helical gear?
... ...

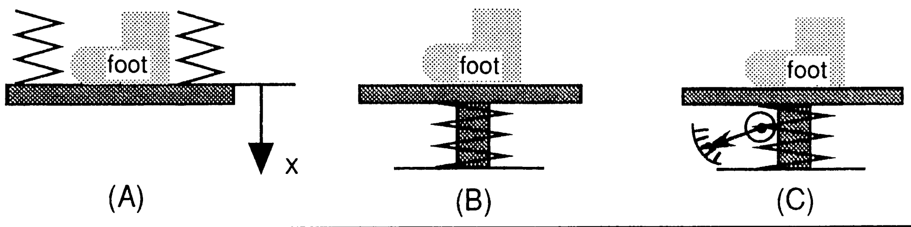


**Figure 16: The Design Experiment Protocol Data**

Figure 18 shows the design specifications. The specifications are described in world *w0*. By performing abduction three times with *rule (1)*, *rule (2-1)* and *rule (2-2)*, the design object is evolved to world *w3* (see Fig. 15). This world can be developed to world *w41* or world *w42* separately by applying *rule (3)* or *rule (4)*. We find that the mechanism described in world *w42* (see Fig. 16 (B)) is easier to realize than the mechanism described in world *w41* (see Fig. 16 (A)), we select world *w42* to continue our design. In world *w6*, there is a contradiction caused by *rule (2-2)* and *rule (6)*. Although this contradiction can be solved by circumscription, this requires a revision of the specifications that we do not want to do. We come back to world *w5* and continue our design work by adding a gear box. In world *w10*, another contradiction caused by *rule (10)* and *rule (1)* is found. This time we solve the contradiction by circumscription (Fig. 19). After the

(1) **IF** weight(W) & canMeasure(S,W) & support(S,W) **THEN** scale(S),
(2-1) **IF** displacement(D) & indictor(I) & has(S,I) & weight(W) & translate(S,W,D)
    **THEN** canMeasure(S,W),
(2-2) **IF** isInProportion(S,W,D) & weight(W) & displacement(D)
    **THEN** translate(S,W,D),
(3) **IF** spring(SP) & push(S,SP) & has(S,SP) & weight(W) **THEN** support(S,W),
(4) **IF** spring(SP) & pull(S,SP) & has(S,SP) & weight(W) **THEN** support(S,W),
(5) **IF** rackAndPin(RP) & has(S,RP) & weight(W) & displacement(D)
    **THEN** isInProportion(S,W,D),
(6) **IF** isInProportion(S,W,D) & weight(W) & displacement(D)
    & >(maxValue[W]/maxValue[D],10) **THEN** ~translate(S,W,D),
(7) **IF** isInproportion(S,W,D) & weight(W) & displacement(D)
    & >(maxValue[W]/maxValue[D],10) & manyGears(MG)
    & hasManyGears(S) **THEN** translate(S,W,D),
(8) **IF** manyGears(MG) & hasManyGears(S) **THEN** ~hasUpwardIndictor(S),
(9) **IF** ~hasUpwardIndictor(S) **THEN** ~easyToSee(S),
(10) **IF** ~easyToSee(S) **THEN** ~scale(S),
(11) **IF** hasUpwardIndictor(S) **THEN** easyToSee(S),
(12) **IF** hasManyGears(S) & helicalGear(HG) & has(S,HG)
    **THEN** hasUpwardIndictor(S),
... ...

**Figure 17: Formalized Knowledge**

**specifications:**
scale(sc1)
=(maxSupportWeight[sc1],100)
=(maxDisplacement[sc1],5)
...

**Figure 18: Design Specifications**

circumscription, *rule (1)* is changed. Because of the revision of the knowledge, the fact *scale(sc1)* cannot be concluded again. So it is focused as a new problem to be solved. We come back to the world *w0* to perform the design from almost the very beginning. This time, because we have some *experience* about the design of the scale, we can evolve the design object smoothly. Also by using a helical gear, it is possible to see the indictor of the scale upward. Figure 20 shows part of the database in world *w20*.

In the design of the gear box, for some reason, a gear pair is used. The related shafts and bearings are also generated. All these data construct a primary model shown in Fig. 8. Then the qualitative reasoning system reasons out all possible behaviors of the design object and build a network of concepts from the primary model (Fig. 9). In the evaluation stage, because we want to evaluate the design object from various points of view, several aspect models are generated. The metamodel describes the relationships among these data

(1) **IF** weight(W) & canMeasure(S,W) & support(S,W) **THEN** scale(S),
(10) **IF** ~easyToSee(S) **THEN** ~scale(S),

**(a) Two Rules with a Contradiction**

(1') **IF** weight(W) & canMeasure(S,W) & support(S,W) & ~ab1 **THEN** scale(S),
(10') **IF** ~easyToSee(S) & ~ab2 **THEN** ~scale(S),

**(b) Adding Abnormal Predicates to Rules**

(priority: ab2 > ab1)
ab1 = ~easyToSee(S)
ab2 = false

**(c) Solutions of Circumscription**

(1') **IF** weight(W) & canMeasure(S,W) & support(S,W) & easyToSee(S)
**THEN** scale(S),
(10') **IF** ~easyToSee(S) **THEN** ~scale(S),

**(d) Modified Rules with the Solution of Circumscription**

**Figure 19: Contradiction Resolution by Circumscription**

```
scale(sc1), easyToSee(sc1), weight(weight:2546),
canMeasure(sc1,weight:2546), support(sc1,weight:2546),
displacement(displacement:11377),
translate(sc1,weight:2546,displacement:11377),
isInProportion(sc1,weight:2546,displacement:11377),
spring(spring:13414), pull(sc1,spring:13414),
rackAndPin(rackAndPin:5480),manyGears(manyGears:2599),
hasManyGears(sc1), hasUpwardIndictor(sc1),
helicalGear(helicalGear:1155), has(sc1,helicalGear:1155),
twoGearsMechanism(twoGearsMechanism:13994), gear(gear:9600),
gear(gear:16252), gearPair(gear:9600,gear:16252),
~sameDirection(gear:9600,gear:16252),
... ...
```

**Figure 20: Part of the Database in a World**

in order to keep the consistency of different aspect models. Figure 21 shows part of the data in the metamodel mechanism concerned about a gear pair.

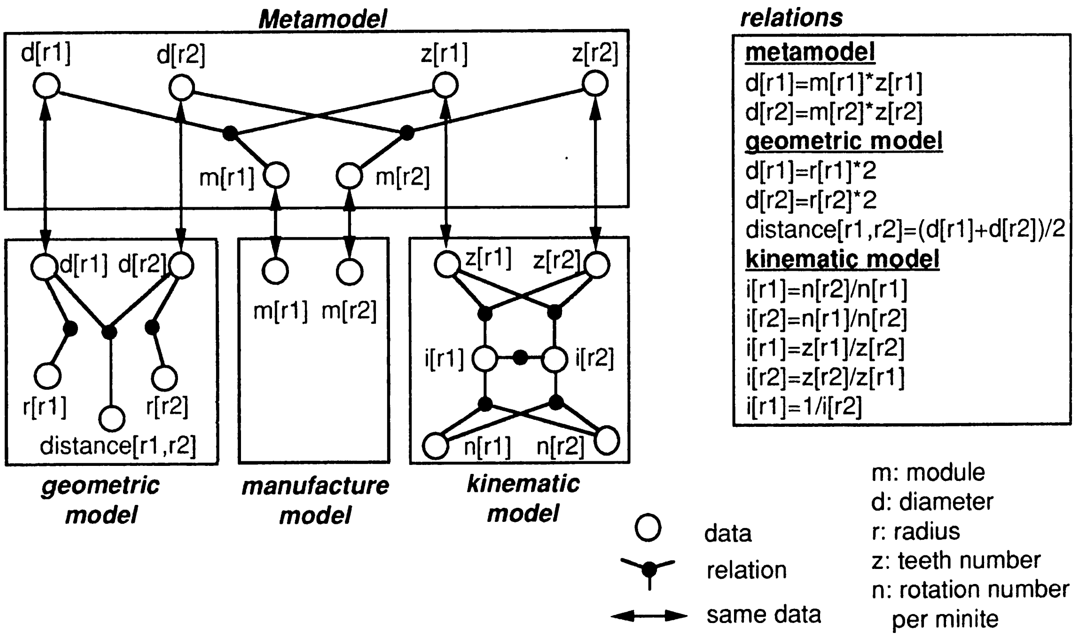


**Figure 21: The Metamodel for a Gear**

## 5. CONCLUSIONS

This paper described a preliminary report about a project to develop a third generation intelligent CAD system, called Intelligent Integrated Interactive CAD, currently conducted at the University of Tokyo. Since design knowledge is considered extremely huge and complex, we put an emphasis on theoretical considerations. This resulted in new techniques for representing design process knowledge, a new framework for representing design object knowledge based on qualitative physics, and as a whole integrated use of design knowledge.

Our results are summarized as follows.

(1) The computable design process model, which is based on the cognitive design process model derived from design experiments, is useful to describe design process knowledge.

(2) A design object should be evaluated from various points of view. The metamodel mechanism was developed to integrate these different aspect models and is used as a design object handling framework for IIICAD. It is based on qualitative descriptions about the design object, i.e., physical features, and represents knowledge about the physical world.

(3) The metamodel mechanism realizes the evolutionary aspect of design which is also a result of theoretical work on design processes. A prototype of IIICAD is developed and it describes two types of design process knowledge, i.e., action level knowledge and object level knowledge. The metamodel mechanism realizes object level

operations.

Future work includes the development of an intelligent user interface, an automatic design knowledge acquisition mechanism, and a large ontological knowledge base.

## ACKNOWLEDGEMENT

We would like to thank students of Yoshikawa-Tomiyama Laboratory of the Department of Precision Machinery Engineering, Faculty of Engineering, the University of Tokyo, for their help and constructive comments to this study.

## REFERENCES

(Bobrow, 1985)
D.G. Bobrow (ed.), *Qualitative Reasoning about Physical Systems,* MIT Press, Cambridge, MA, USA, 1985.

(Brown and Chandrasekaran, 1985)
D.C. Brown and B. Chandrasekaran, ''Expert Systems for a Class of Mechanical Design Activity'', in *Knowledge Engineering in Computer-Aided Design, Proceedings of the IFIP W.G. 5.2 Working Conference 1984 (Budpest),* J.S. Gero (ed.), North-Holland, Amsterdam, 1985, pp. 259-290.

(Cutkosky and Tenenbaum, 1990)
M.R. Cutkosky and J.M. Tenenbaum, ''Research in Computational Design at Stanford'', *Research in Engineering Design,* 2(1), 1990, pp. 53-59.

(Dixon and Cunningham, 1989)
J.R. Dixon and J.J. Cunningham, ''Research in Design with Features'', in *Intelligent CAD, I,* H. Yoshikawa and D.C. Gossard (eds.), North-Holland, Amsterdam, 1989, pp. 137-148.

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

(Forbus, 1984)
K.D. Forbus, ''Qualitative Process Theory'', *Artificial Intelligence,* 24(3), North-Holland, 1984, pp. 85-168.

(Gero, 1987)
J.S. Gero (ed.), *Expert Systems in Computer-Aided Design, Proceedings of the IFIP W.G. 5.2 Working Conference 1987 (Sydney),* North-Holland, Amsterdam, 1987.

(Goldberg and Robson, 1983)
A. Goldberg and D. Robson, ''Smalltalk-80: The Language and its Implementation'', Addison-Wesley, Reading, MA, USA, 1983.

(Hayes, 1985)
P.J. Hayes, ''Naive Physics Manifesto I: Ontology for Liquids'', in *Formal Theories of the Commonsense World,* J. Hobbs and R. Moore (eds.), Ablex Publishing, Co, Norwood, New Jersey, 1985, pp. 71-107.

(Inui and Kimura, 1990)
M. Inui and F. Kimura, ''Representation and Manipulation of Design and

Manufacturing Processes by Data Dependency'', in *Intelligent CAD, II,* H. Yoshikawa and T. Holden (eds.), North-Holland, Amsterdam, 1990, pp. 183-201.

(Johnson-Laird, 1983)
P.N. Johnson-Laird, *Mental Models,* Cambridge University Press, Cambridge, UK, 1983.

(Kiriyama, Tomiyama, and Yoshikawa, 1990)
T. Kiriyama, T. Tomiyama, and H. Yoshikawa, ''Qualitative Reasoning and Conceptual Design with Physical Features'', in Proceedings of the 4th International Workshop on Qualitative Physics, Lugano, Switzerland, 1990, pp. 153-160.

(Kiriyama, Yamamoto, Tomiyama, and Yoshikawa, 1989)
T. Kiriyama, F. Yamamoto, T. Tomiyama, and H. Yoshikawa, ''Metamodel: An Integrated Modeling Framework for Intelligent CAD'', in *Artificial Intelligence in Design,* J.S. Gero (ed.), Computational Mechanics Publications, Southampton, Boston, 1989, pp. 429-449.

(de Kleer, 1986)
J. de Kleer, ''An Assumption-based TMS'', *Artificial Intelligence,* 28, 1986, pp. 127-162.

(Lenat and Guha, 1989)
D.B. Lenat and R.V. Guha, *Building Large Knowledge-base Systems: Representation and Inference in the Cyc Project,* Addison-Wesley, Reading, MA, USA, 1989.

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

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

(Nakagawa and Mori, 1987)
H. Nakagawa and T. Mori, ''Computable Circumscription in Logic Programming'', *Transactions of Information Processing Society of Japan,* 28(4), pp. 1987, 330-338, in Japanese.

(Suzuki, Ando, and Kimura, 1990)
H. Suzuki, H. Ando, and F. Kimura, ''Synthesizing Product Shapes with Geometric Design Constraints and Reasoning'', in *Intelligent CAD, II,* H. Yoshikawa and T. Holden (eds.), North-Holland, Amsterdam, 1990, pp. 309-324.

(Takeda, Tomiyama, and Yoshikawa, 1990)
H. Takeda, T. Tomiyama and H. Yoshikawa, ''Logical Formalization of Design Processes for Intelligent CAD Systems'', in *Intelligent CAD, II,* H. Yoshikawa and T. Holden (eds.), North-Holland, Amsterdam, 1990, pp. 325-336.

(Takeda, Veerkamp, Tomiyama, and Yoshikawa, 1990)
H. Takeda, P.J. Veerkamp, T. Tomiyama, and H. Yoshikawa, ''Modeling Design Processes'', in *AI Magazine,* 11(4), 1990, pp. 37-48.

(Tomiyama, 1989)
T. Tomiyama, ''Object Oriented Programming Paradigm for Intelligent CAD

Systems'', in *Intelligent CAD Systems II: Implementational Issues,* V. Akman, P.J.W. ten Hagen, and P.J. Veerkamp (eds.), Springer-Verlag, Berlin, 1989, pp. 3-16.

(Tomiyama and ten Hagen, 1987)
T. Tomiyama and P.J.W. ten Hagen, ''The Concept of Intelligent Integrated Interactive CAD Systems'', CWI Report No. CS-R8717, Centre for Mathematics and Computer Science, Amsterdam, 1987.

(Tomiyama, Xue, and Ishida, 1991)
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,* P.J.W. ten Hagen and P.J. Veerkamp (eds.), Springer-Verlag, Berlin, Forthcoming.

(Tomiyama and Yoshikawa, 1987)
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),* H. Yoshikawa and E.A. Warman (eds.), North-Holland, Amsterdam, 1987, pp. 95-130.

(Umeda, Takeda, Tomiyama, and Yoshikawa, 1990)
Y. Umeda, H. Takeda, T. Tomiyama, and H. Yoshikawa, ''Function, Behaviour, and Structure'', in *Applications of Artificial Intelligence in Engineering V, Vol 1: Design, Proceedings of the Fifth International Conference,* Boston, USA, J.S. Gero (ed.), Springer-Verlag, Berlin, 1990, pp. 177-193.

(Veerkamp, 1989)
P.J. Veerkamp, ''Multiple Worlds in an Intelligent CAD System'', in *Intelligent CAD, I,* H. Yoshikawa and D.C. Gossard (eds.), North-Holland, Amsterdam, 1989, pp. 77-88.

(Veth, 1987)
B. Veth, ''An Integrated Data Description Language for Coding Design Knowledge'', in *Intelligent CAD Systems I: Theoretical and Methodological Aspects,* P.J.W. ten Hagen and T. Tomiyama (eds.), Springer-Verlag, Berlin, 1987, pp. 295-313.

(Weld and de Kleer, 1990)
D.S. Weld and J. de Kleer (eds.), *Readings in Qualitative Reasoning about Physical Systems,* Morgan-Kaufmann Publishers, Inc., San Mateo, CA, 1990.

(Yoshikawa, 1981)
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),* T. Sata and E.A. Warman (eds.), North-Holland, Amsterdam, 1981, pp. 241-253.

## DISCUSSION:
### INTELLIGENT INTEGRATED INTERACTIVE CAD: A PRELIMINARY REPORT

SEVENLER: I was very excited to read the proceedings of this group in 1980. I remember reading the papers of Professors Yoshikawa and Tomiyama. Now after about 10 years, the title of this paper is still "A Preliminary Report". When do you think you will have commercially useful results?

XUE: It is very difficult to say when Intelligent CAD can be used for commercial uses, but I think that the concept of Intelligent CAD is very important. About 30 years ago the concept of CAD was proposed, and they said they could do design in the near future. Then, a long time passed but there was no improvement. So there was a lot of criticism about CAD at that time.

Now we can't say when Intelligent CAD will be available for commercial use, but I think Intelligent CAD is very important and it will be used commercially in the near future.

SEVENLER: The examples I see in this conference are still very simple. I remember Dave Brown's work on air cylinder design years ago; now it is relief valve design! You are right that we need lots of time, but is there a plan so that progress we have made in the last 10 years and progress we will make in the next 10 years can head towards some results which would be commercially useful?

BIJL: Perhaps we should address this in the general discussion session.

NIELSEN: I don't get a feeling of how bounded your problem is. It seems large. I was wondering if you could tell me what domain of designs or design processes this is restricted to?

XUE: This IIICAD system proposed a framework to organize the information of the design object and the design processes. So it is just a framework. What kind of problem this system can solve depends on what kind of knowledge we describe.

NIELSEN: Could you give me an example of something you couldn't do with this framework and something you might design?

XUE: For example, I can show you the design of a scale. We have a knowledge base for that. Maybe it is a simple example, but at this stage, we just want to show how the Intelligent CAD system should be constructed.

VELTKAMP: I thought you were already working on incorporating qualitative physics into the IIICAD system. Did you make any progress?

XUE: Yes. The qualitative physics in this system is used to describe the ontological knowledge, the deep knowledge which is the knowledge about the physical world.

First we describe the design process knowledge in the scenarios. A scenario is a kind of set of production rules. By using the production rules, we get a primary model. We can use the qualitative physics to explain this model, such as to generate a metamodel. It is a kind of concept network used to evaluate the original idea.

**VELTKAMP**: Were you setting up large knowledge bases with physical features?

**XUE**: Yes. At our laboratory, we are now building up a large knowledge base. It is based on physical features. We have built about 2000 physical features.

**VELTKAMP**: Can you use that already? Do you have results?

**XUE**: We have some problems that should be solved in the near future.

**BROWN**: I like the idea of building frameworks. I think it will raise many interesting questions. In your framework, how much of the design process is done automatically and how much is done by the person? For example, when decomposing functions to sub functions it wasn't clear whether the system provides suggestions.

**XUE**: As a matter of fact, we have several methods described in the action level knowledge base to make a suggestion, that is, action level scenarios. They tell that there are several ways to make our suggestions, such as abduction, a kind of backward reasoning, or by the FBS Modeler.

It depends on how the user uses it. If the problem can be solved by the object level scenarios, we use them. If it can be solved by the FBS Modeler, then we use it. We have several ways to make a suggestion to one program.

**BROWN**: But does the user have to decide which of these techniques to pick?

**XUE**: Yes.

**GOEL**: I am a little concerned about your use of the term "abduction". Abduction is a kind of inference to the best explanation. My understanding is that it is connected with the idea of forming a reasonable, a good, or best explanation for some data set. But you seem to be equating abduction with backward reasoning. That may well be right, but I am just a little concerned about that.

**XUE**: There is a lot of other work concerned with abduction in our laboratory. I don't think that abduction can be performed only by backward reasoning. Backward reasoning can be used to perform abduction; it's the simplest way.

**GOEL**: But backward reasoning is not identical to abduction. Not all backward reasoning is necessarily abductive.

Moving to the point that Eric Nielsen raised: would it be fair to say that one way of characterizing the class of domains to which your theory is applicable is the domains in which the types of knowledge you are assuming are in fact available?

For instance, you are assuming that you have some function/behavior/structure model available so that you can connect function to behavior and behavior to structure.

But it is not clear that such models are in fact available in all design domains. One design domain that comes to mind in which such models may be hard to find is architectural design. It is not clear to me that for architectural design such global function/behavior/

structure models are available.

Would that be one way of characterizing the class of domains to which your theory might be applicable?

**XUE**: Yes. It gives a framework for how design processes and objects can be organized by the system. As to the special aspects, such as architecture design, mechanical design or electronic design, we must consider specific properties of this design. So it depends on how you build your knowledge base.

**MacCALLUM**: I am interested in the methodology you are proposing for maintaining consistency among different aspects of the model. Do you use a truth maintenance system?

**XUE**: Yes.

**MacCALLUM**: Is it domain dependent or domain independent?

**XUE**: Domain independent.

**MacCALLUM**: What is it that controls its activation? What is it that makes the truth maintenance system work? Is it any change, or a group of changes, or what?

**XUE**: In our system, as a matter of fact, there are two ATMS. One is in the primary model and one is in the metamodel. The primary model is built up by the inference of the design process handling system. The suggestions can be considered as assumptions in the ATMS. So when we perform the operation of abduction, we get a suggestion which can be considered as an assumption in the ATMS. So the other data that are derived from the assumptions have relationships with the assumptions.

So if a contradiction occurs, we can find out whether the assumptions are contradicted. We can remove part of the assumptions in order to keep the integration of the data.

**FALTINGS**: The examples you described in your paper are mechanisms. In mechanisms one important aspect is that of geometry. Parts interact by the way they are arranged in space. When you deal with such interactions, the relationship between functional or behavioral and physical features can become much more complex than what you seem to allow in your models.

In cases where you have spatial interaction, the relationship between the functional and behavioral features and the physical features of your artifact is not really a mapping, one to one, but can be a very complicated relationship involving non-monotonic dependencies. I think in your CAD system you would have difficulty handling that. Do you have an ideas to extend it?

**XUE**: At this stage, we just have some relationship to the function and behaviors. Then we can decompose the functions to several sub functions, and then we can also realize the functions by the physical features. But how to select physical features is a problem.

**KANNAPAN**: Figure 4 has action level inference and object level inference. You described knowledge of actions. What's going on in the top box, the Ka, C and O?

**XUE**: The "Ka" is the action level knowledge to describe the general knowledge on the design actions. "Ko" is the object level knowledge to describe the relationships between the design objects and properties of the design objects.

We added a design specification as P which is considered as a "problem". Then because the action level design process knowledge described the relationships between the status of the database and possible operations, this inference gives users the possible operations.

**GRABOWSKI**: Could you give an example for general knowledge?

**KANNAPAN**: Specifically, what is K? What is C?

**XUE**: "C" is the condition. It is the condition of the object level database. And "O" is the operation to the object level database.

For example, this is an action level scenario used to describe action level knowledge. First all of the rules should be matched, and then the successful rules can be shown by a menu to the user. If we have a suggestion, then this rule can succeed. Then we can make a development or an evaluation or a conclusion with it.

So we then select one of these operations to perform the design. The operations can be the selections of the object level scenarios or the object level inference such as abduction, deduction, or circumscription.