Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

T. Tomiyama, P.J.W. ten Hagen

Representing knowledge in two distinct descriptions:
extensional vs. intensional

Computer Science/Department of Interactive Systems Report CS-R8728 June

The Centre for Mathematics and Computer Science is a research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a nonprofit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O.).

Representing Knowledge in Two Distinct Descriptions: Extensional vs. Intensional

Tetsuo Tomiyama, Paul J.W. ten Hagen
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

This paper describes a theory of knowledge on which future CAD systems can stand. First, we present two distinct description methods, viz. extensional and intensional. Second, these two are compared in the context of CAD applications and their advantages and disadvantages are clarified. Finally, we propose a new data description method which combines extensional and intensional description methods.

1982 *CR Categories:* H.1.2, H.2.1, I.2.4, J.6

Key Words & Phrases: conceptual modeling, data modeling, knowledge engineering, knowledge representation, CAD.

Note: This report will be submitted for publication elsewhere.

1. Introduction

Developing intelligent CAD (Computer Aided Design) systems is crucial in order to achieve high productivity and better quality products. Knowledge engineering is considered one of the key issues [8, 9]. In developing such systems we must be aware that designing is a highly intellectual activity. This requests a clean, sound, and robust theoretical basis to capture design knowledge.

For this purpose, we have been involved in establishing a *theory of CAD* as a part of the IIICAD (Intelligent Integrated Interactive CAD) project [15, 17, 18]. Since there are two aspects in designing, namely, design objects and design processes, we need one theory to describe design objects and the other to formalize design processes. The latter might be called *design theory* [20] and it can be domain independent. The former might be called *theory of design objects* and is domain dependent. In case of VLSI this must be "VLSI theories" and should be replaceable by "mechanical engineering theories" if we need CAD systems for machine design. In addition to these two, there might be more general theories as long as we deal with physical world. Perhaps we need theories about the underlying principle of the world, e.g., *naive physics* [3].

Knowledge engineering as a technique to describe design knowledge is probably regarded as software developing methodology and environment [4, 18]. Although it tells what kind of techniques to use, it does not tell how to describe the knowledge. For example, even if we are given a knowledge engineering tool, e.g., a frame system [13], we still have difficulties in codifying knowledge: *What comes as a frame? What comes as a slot?* This problem is relevant to the problem of knowledge acquisition and few solutions have been proposed. We realize that we should have a theory to describe knowledge in a given scheme. We call it *theory of knowledge* and this paper is a first approach to it.

Report CS-R8728
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

In CHAPTER 2 we begin with formalizing our recognitions of the world. We shall see that there are entities and their properties. Then we shall acknowledge two distinct description methods, an extensional description and an intensional one, and follow with a comparison of them.

CHAPTER 3 discusses CAD applications. We categorize logical manipulations into two categories; add and renewal operations. In extensional descriptions, data operations that are important for CAD applications can be implemented by add operations, while intensional descriptions are implemented by renewal operations. Add operations are cheaper than renewal operations in terms of performance. This implies that CAD data structures should be based on extensional descriptions.

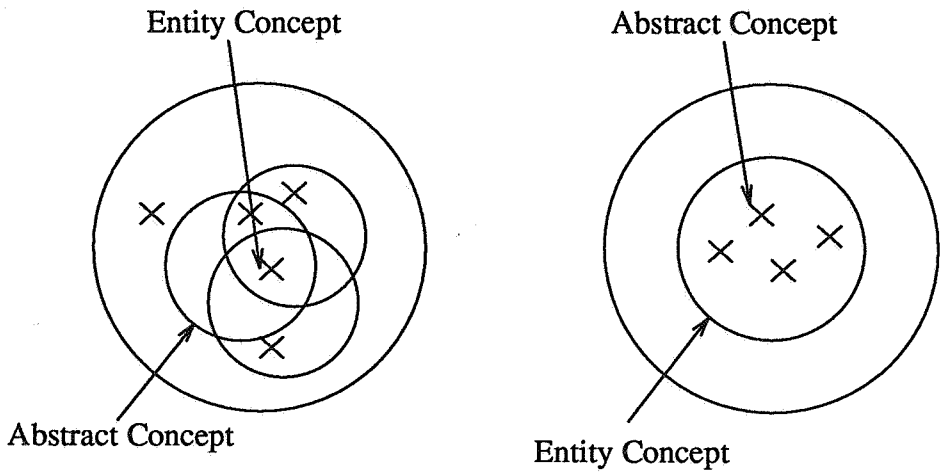
However, for simple inquiries, an intensional description is more convenient than an extensional one. We, therefore, propose in CHAPTER 4 a new data description method to take advantage of both extensional and intensional descriptions. This new approach is employed in the design of IDDL (Integrated Data Description Language), the kernel language of IIICAD [18].

2. Description of the World

In this chapter we begin with a metaphysical discussion in order to clarify two important and opposing concepts in describing knowledge in general. Next, we try to formulate those two concepts mathematically and compare them, so that we can identify their differences. We shall also find outt that they are equivalent in principle.

2.1. Entities, Attributes, and Relationships

It may be reasonable to start the discussion by admitting that in our world we have *entities* such as a dog, the Sun, a shaft, etc. These entities might be *categorized* or *classified* by observation and abstraction. As a result of categorization we will have a group of entities which have common nature or behavior. By doing so, we recognize a *relationship* which characterizes that group and we form an *abstract concept*. This further results in the concept of *attributes*, such as weight, length, color, etc., which may or may not have *values* and which are *attributed* to entities. We can introduce secondary relationships and attributes as well; i.e., relationships among relationships, relationships among attributes, and attributes of relationships. This recognition of the world has been dominant for many centuries in history.



(a) Extensional Description **(b) Intensional Description**

FIGURE 1. Two opposing ways of describing the world

Next we formalize this perception mathematically. *Set notation* has been the most basic and powerful tool among others. In set theory, elements of a set are treated, more or less homogeneously, as symbols which has no inner structure. Elements of a set are "equal" and there is no way to compare or modify elements; we can only create and remove elements. On the other hand, subsets (or topology, base, etc.) can be manipulated by set operations such as conjunction, disjunction, negation, etc. Hence subsets can be compared and modified.

We notice two opposite ways for describing the world [16] as shown in FIGURE 1.

(1) **Extensional description**: Entities are primary (FIGURE 1 (a)). Therefore, entities are the elements, and attributes and relationships form the topology of the entity set.

(2) **Intensional description**: Abstract concepts are primary (FIGURE 1 (b)). Therefore, relationships and attributes are the elements, and entities form the topology of the world set.

In our view, distinguishing these two description methods is the key to solve the knowledge representation problem. In the following sections, we will examine and compare them more precisely.

2.2. Extensional and Intensional Descriptions

From now on, we use the words *entity concept* instead of entity and *abstract concept* instead of attribute and relationship, because we are going to discuss concepts, not the real world.

2.2.1. Extensional Description

An *extensional description* is defined as a situation where an entity concept is an element of the entity concept set and abstract concepts are its topology (FIGURE 1 (a)). This description method has the following properties:

(1) Entities are primary. They are dealt with just as symbols and have no meaning other than just being symbols. There is no predefined attributes inside an entity and it is impossible to decompose entities into smaller parts. This means that the extensional description is holistic, implying that the world is built only from a collection of facts about relationships among (symbolic) entities.

(2) Mathematically, this situation is described as follows. Let e_i and A_j be an entity concept and an abstract concept, respectively. Then, A_j is defined extensionally, by

$$A_j \equiv \{e_1, e_2, \cdots\}.$$

(3) This definition suggests that first a common property (or relationship) is found by an observation and then named A_j. In other words, an abstract concept A_j is stating a relationship between entities, e_i.

(4) Descriptions of an entity concept, therefore, consist of facts each of which is dependent on other entities. In this sense, an extensional description is relative.

2.2.2. Intensional Description

An *intensional description* is defined to be a situation where an abstract concept is an element of the abstract concept set and entity concepts are its topology (FIGURE 1 (b)). This description method has the following properties.

(1) Attributes and relationships are primary, which means they are dealt with as symbols which have no meaning other than being symbols. On the other hand, entities are considered to be somewhat structural, because they are described by predetermined abstract concepts. This further suggests that an entity is decomposable into smaller parts (reductionism).

(2) The situation in FIGURE 1 (b) is defined mathematically as follows. Let a_i and E_j be an abstract concept and an entity concept, respectively. Then, using a_i, E_j is defined intensionally by

$$E_j \equiv \{a_1, a_2, \cdots\}.$$

(3) An entity concept in an intensional description is generated from abstract concepts which are predefined, hence absolute. An intensional description is absolute.

(4) An entity concept can be often described by a fixed number of abstract concepts as a Cartesian product set, like

$$E_j \equiv \{(a_1, a_2, \cdots) \mid \Sigma(a_1, a_2, \cdots)\},$$

where Σ indicates additional constraints. In an intensional description, an entity concept is equivalent to a collection of attributes.

(5) In summary, an intensional description embodies predefined concepts (*intensions*) which must be explicitly expressed in an extensional description.

2.3. Hierarchical Example

Let us compare an extensional description and an intensional description. In the following discussion we employ predicate logic instead of set notation; yet we are not trying to discuss the descriptive power of predicate logic.

Consider a car which consists of lots of parts, from an engine to a radio. The engine itself can be decomposed into thousands of smaller parts. We have traditionally been calling these part-assembly relationships a *hierarchy* (FIGURE 2).

2.3.1. Extensional Description of the Example

In an extensional description method, the most natural interpretation for FIGURE 2 is as follows:

(1) First we regard a symbol, such as *CAR*, as an atom which cannot be further decomposed. Thus, *CAR* is an identifier and can be replaced by any character string such as *@AB5306*.

(2) A line that connects two symbols is regarded as the relationship between them. We can use the binary relationship, *has*, (or *belongs–to* which is the other way around) for the time being, although its meaning cannot be directly defined at the moment.

(3) For example the relationship between *CAR* and *ENGINE* will be interpreted such that *CAR* and *ENGINE* make a subset called *has*, i.e.,

$$has = \{CAR, ENGINE\}.$$

(4) Because the relationship *has* is specific to *CAR* and *ENGINE*, we may want to introduce as many names as the relationships. However, this is not necessary; we can use the same name for all other relationships, as far as we understand that implicitly the meaning of these relationships are the same.

(5) Therefore, this hierarchy will be denoted as follows:

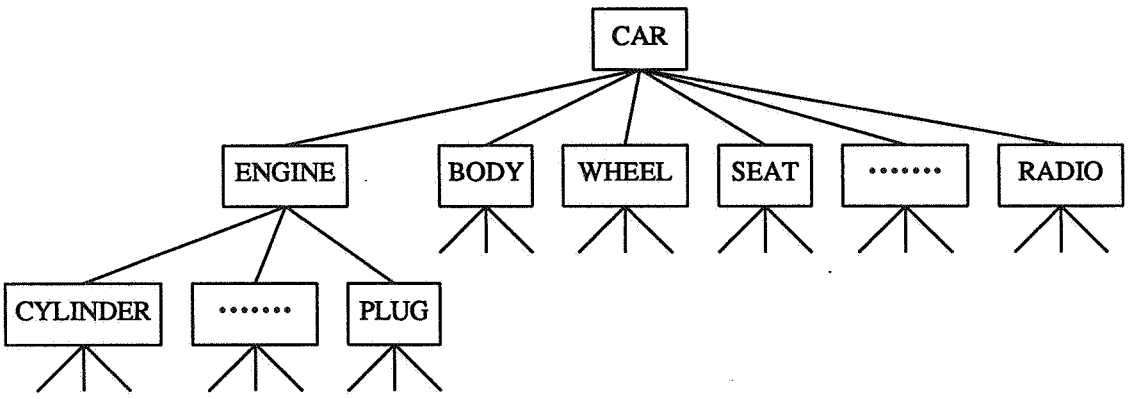


FIGURE 2. An abstraction of a car

has(*CAR*, *ENGINE*),
has(*CAR*, *BODY*),
has(*CAR*, *WHEEL*),
has(*CAR*, *SEAT*),
has(*CAR*, *RADIO*),
has(*ENGINE*, *CYLINDER*),
has(*ENGINE*, *PLUG*).

2.3.2. Intensional Description of the Example

In an intensional description method, FIGURE 2 will be interpreted in a totally different way:

(1) In this description method, there is an underlying belief that a *CAR* is decomposable into fragmental parts such as *ENGINE*, *BODY*, etc. A *CAR* is built from those parts.

(2) A symbol, such as *CAR*, is not a mere identifier. It has an inner structure which can be further decomposed. Therefore, an intensional description method may lead us to the idea of *typing* or *classes*.

(3) The relationship *has* of SECTION 2.3.1. is hidden in the structure of entities.

(4) The following is an intensional notation of this hierarchy:

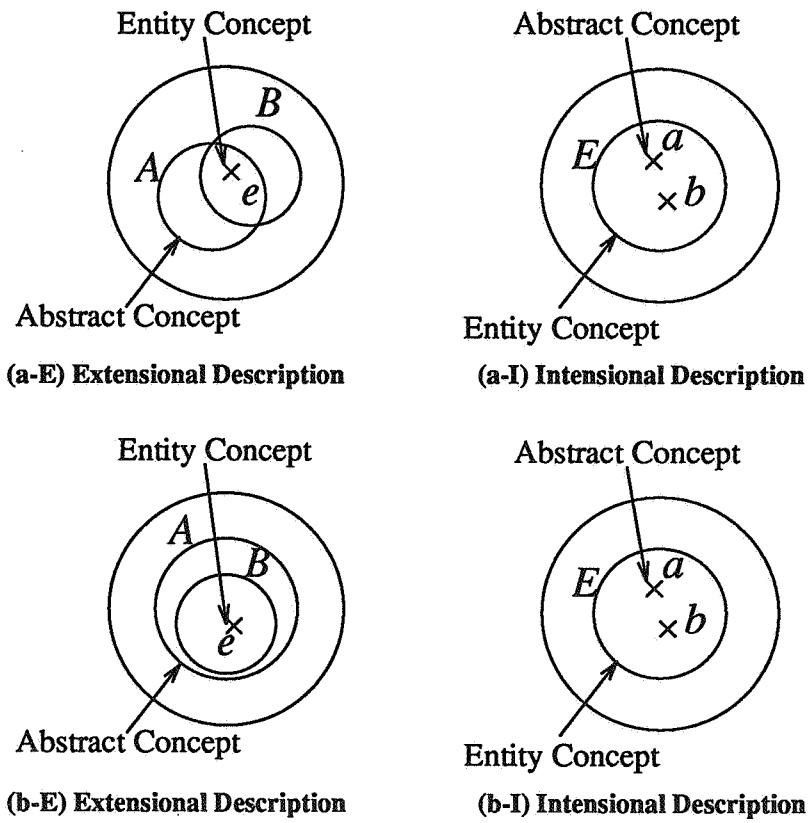


FIGURE 3. Comparison of extensional/intensional descriptions

$CAR = (ENGINE, BODY, WHEEL, SEAT, \cdots, RADIO)$
$ENGINE = (CYLINDER, \cdots, PLUG)$

2.4. Some Problems

2.4.1. Rigidity of Intensional Descriptions

Comparing the extensional and intensional descriptions of the previous sections, we can detect problems with an intensional description. Consider $CAR = (ENGINE, BODY, WHEEL, SEAT, \cdots, RADIO)$. We might assume the second term of this expression refers to the body concept no matter what identifier comes. It might be hard to insert a new term between *ENGINE* and *BODY*.

Intensional descriptions are, therefore, inflexible and rigid.

2.4.2. Data Deterioration in Intensional Descriptions

FIGURE 3 shows another disadvantage of the intensional description method. FIGURE 3 (a-E) shows that two different abstract concepts are denoting an entity. This situation can be also identically depicted in the intensional description as in FIGURE 3 (a-I). However, as in FIGURE 3 (b-E), if two similar or hierarchical abstract concepts are denoting an entity, the similarity or the hierarchy cannot be expressed so well in the intensional description (see FIGURE 3 (b-I)); they are just represented in the same way as in FIGURE 3 (a-I).

Thus, in case of the intensional description method, slight differences in meaning would be lost or ignored and similar concepts would be recognized differently.

2.4.3. Information Loss in Data Exchange between Extensional and Intensional Descriptions

As pointed out in SECTION 2.2.2, an intensional description embodies predefined concepts which must be explicitly expressed in an extensional description. Because of these predefined concepts, data exchanges among different CAD systems might result in loss of information when both intensional and extensional descriptions are involved.

Let us consider a plane s in three-dimensional space defined by three points, or by a normal vector and a point, or alternatively by two vectors originating from the same point. We have two ways of representing this fact:

(1) In an intensional description, s will be represented by three points, p_1, p_2, and p_3. In a relational database system, this is expressed by a tuple $plane(p_1, p_2, p_3)$. Note that s will not explicitly appear in this tuple.

(2) In an extensional description, we use the facts that s has p_1, s has p_2, and s has p_3. Relations for this case can be three tuples $PLANE(s)$, $POINT(p)$, and $HAS(s, p)$, and the entire fact will be represented by

$$\forall s[PLANE(s)]\exists p_1, p_2, p_3\ [POINT(p_1), POINT(p_2), POINT(p_3), p_1 \neq p_2, p_2 \neq p_3, p_3 \neq p_1, HAS(s, p_1), HAS(s, p_2), HAS(s, p_3)].$$

Note that we can think of s regardless of p_1, p_2, or p_3; i.e., to define s, we do not need to know its intension.

Now, we can point out a problem. It is possible to define a plane s intensionally both by three mutually different points, p_1, p_2, and p_3, and by a normal vector v and a point p. Suppose we have a CAD system A which employs the former description, B which uses the latter description, and C which uses an extensional description method (see FIGURE 4). In system A, s will be described by

$$plane(p_1, p_2, p_3),$$

and in system B by

$$plane(p, v).$$

In system C, it will be described extensionally as

$$PLANE(s), POINT(p), HAS(s, p), VECTOR(v), DEFINED\text{-}BY(s, p_1, v), v = (p_2 - p_1) \times (p_3 - p_1)$$

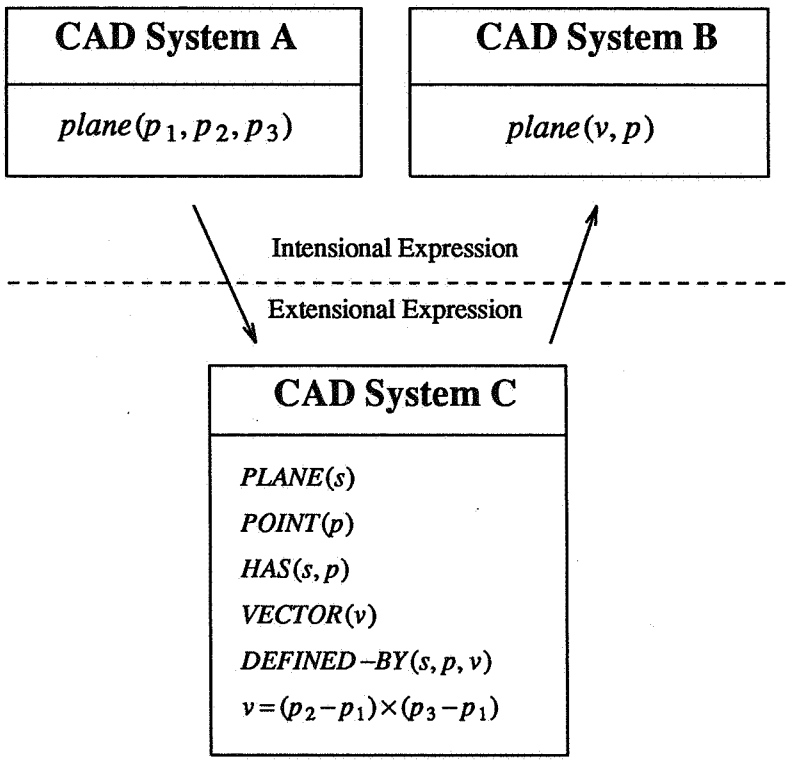


FIGURE 4. Data exchange between two CAD systems

to make it possible to transfer data from A to B via C. In this transformation we have completely lost information about p_2 and p_3. This means there will be a loss of information and inevitable confusion in data exchange shown in FIGURE 4, if there are two different intensional descriptions and an extensional description.

This problem is caused by a fundamental problem in conversion between intensional and extensional descriptions. The intensional descriptions of the systems A and B assume predefined concepts C_a and C_b. Between these two there is an incompatibility which causes the problem. A mathematical explanation is given below. Let $\downarrow x$ be an intension of x, and $\uparrow x$ be an extension of x. Then $\uparrow\downarrow x = x$ always holds. Conversely $\downarrow\uparrow x = x$ does not always hold. A famous example is *the morning star* and *the evening star*. The morning star has an extension, *Venus*. The evening star is an intension of Venus. Thus, an intension of an extension of an entity is not always identical to the entity.

3. CAD Applications

In this chapter, we discuss what kind of data operations and data description methods are required for CAD systems. We compare extensional and intensional data descriptions in the context of CAD applications.

3.1. Characteristics of CAD Applications

CAD applications request several considerations about the data description method, because they are different from business applications in many respects [7, 11]. The followings are some important differences which effect the data description method:

(1) *Diversity*: There are many ways of representing the design objects, e.g., machinery. Those representations are called models or views. Although they represent the identical object, syntactically they are different from each other.

(2) *Dynamic changeability*: Models are changing dynamically during the design process from a vague initial one to detailed final drawings.

(3) *Bulkiness*: Usually, amount of information used in a CAD system is enormous.

(4) *Integrity and consistency*: Despite the diversity and dynamic changeability of models, we need to keep integrity and consistency of the information. This means that any change should propagate through all the models and be inherited from an earlier version to later versions.

(5) *Long transaction*: Compared with business applications, transactions take a longer time; for instance, from one day to a couple of years. Because of multi-user access this becomes a problem.

(6) *Multimedia*: Information is of multimedia nature; it will include text information, numerical information, graphical information, and so on.

Our new data description method for future CAD systems should satisfy these requirements (see CHAPTER 4).

3.2. Data Operation in CAD Systems

In this section, first we categorize operations used in predicate logic. We will see that there are two types, add and renewal. Second, we compare data operations of conventional CAD and future CAD to discuss the nature of data operations specific to CAD systems. From this discussion, we see that extensional data description methods have good properties for CAD applications.

3.2.1. Add and Renewal Operations

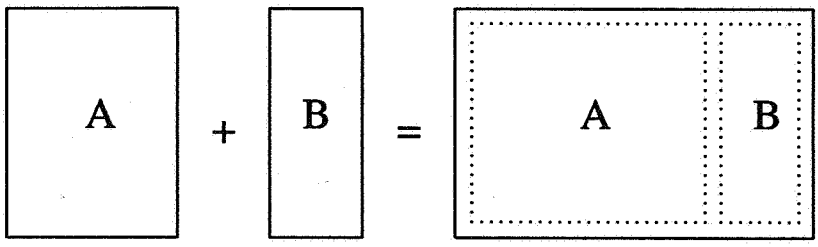
Let us consider data operations in predicate logic (TABLE 1). Except for *query* operations, those operations may change the state of the knowledge base (or database). For example, *assertions* add new facts to the knowledge base. Creation of terms simply adds new objects to the system, which may result in additional assertions about those terms. On the other hand, modification of predicates may result in rewriting the whole predicate system.

From this point of view, we can classify operations into two categories: *Does it rewrite the predicate system or not? Is it a mere addition/removal of facts?* The *"Change?"* column of TABLE 1 answers to these questions. Following this categorization, let us call logical operations which simply add or remove facts *add* operations and those which may rewrite the knowledge *renewal* operations.

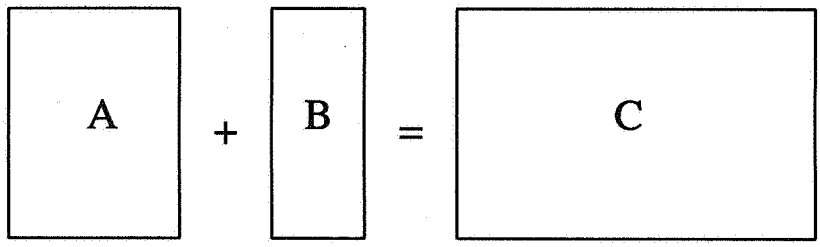
FIGURE 5 illustrates these two types of operations. Add operations simply add new facts to the database and do not have any side effect on the existing parts, whereas renewal operations reorganize of the

TABLE 1. Classification of operations in predicate logic

Object	Operation	Example	Change?	Classification
formula	assertion	$assert(p(a, b))$	No	add
	deletion	$retract(p(a, b))$	No	add
	query	$?p(a, b)$	-	-
predicate	creation	$q(X) :- \cdots$	Yes	renewal
	deletion	$retractall(q)$	Yes	renewal
	modification	$q(X) \rightarrow q'(X)$	Yes	renewal
term	creation	$() \rightarrow t_1, t_2, \cdots$	No	add
	deletion	$t_1, t_2, \cdots \rightarrow ()$	No	add
	modification	$t_1 \rightarrow t_2$	No	add
	copy	$t_1 \equiv t_2$	No	add
	instantiation	$X \equiv t$	-	-



(a) Add operation



(b) Renewal operation

FIGURE 5. Add operations and renewal operations

database. Apparently, renewal operations are much more expensive than add operations. For instance, terms in predicate logic are ordered, which represents the semantics for us; i.e., they have strong mutual dependencies. Changing a predicate structure requires changing the constraints Σ as well (see SECTION 2.2.2). It might even be possible that renewal operations cannot be done automatically. Consequently, if it is possible, we would better avoid those expensive operations.

3.2.2. Data Operations in Conventional and Future CAD Systems

In most of the conventional CAD systems, the following operations are possible.

- Creation of elements (e.g., points, lines, surfaces, volumes, and parts)
- Deletion of elements
- Copying of elements
- Modification of elements
- Simple queries

For instance, moving a line is classified as modification of elements. Note that in conventional systems queries are possible, as far as they are so-called *yes/no* questions and the simplest *what* questions. General *what/why/how* questions are unfortunately impossible. Thus, we can ask questions like:

What is the name of this part? (An answer might be *P0013*.)
Is this line crossing that surface? (An answer might be YES.)

Future CAD systems will be realized based on knowledge engineering techniques as pointed out in CHAPTER 1. In knowledge engineering, the basic operation is *pattern matching*. (Maybe, unification, instantiation, and backtracking are more fundamental, but to do unification pattern matching is indispensable. We do not mention instantiation and backtracking here, because they are not relevant to knowledge representation.) For example, if we want to design a machine quite similar to another designed in the past,

we must be able to choose the most similar candidate from a database. Thus, the following operations based on pattern matching will be necessary (in addition to operations available in conventional CAD systems):

- Comparison of elements based on pattern matching
- Complex query based on pattern matching

Among the above mentioned operations, query and comparison operations do not actually modify the knowledge; however, the remaining operations change the state of the knowledge. We can categorize these operations into two categories, i.e., add operations and renewal operations discussed in SECTION 3.2.1.

In an intensional description these operations (i.e., creation, deletion, copying, and modification operations) are all renewal operations, because in this case entities are predicates which are actually changed. On the other hand, in an extensional description, all of these operations are categorized as add operations, since predicates which denote abstract concepts will not be changed. This is an interesting difference between an extensional description and intensional description methods.

3.3. Data Modeling in CAD Systems

Consider the cube shown in FIGURE 6 and its data modeling methods. We will see in this section that they are in fact based on intensional description methods. Next a data modeling of the same cube based on an extensional description method will be presented. Finally, we shall compare these two description methods and discuss their problems in the context of CAD applications.

3.3.1. Data Modeling in Conventional CAD Systems

In three-dimensional solid modeling, hierarchical modeling techniques are often used (e.g., FIGURE 5). The hierarchy may consist of part, volume, surface, edge, and vertex levels. At each level there are loops which represent relationships in that level; for example, edge information is predefined such that an edge has two

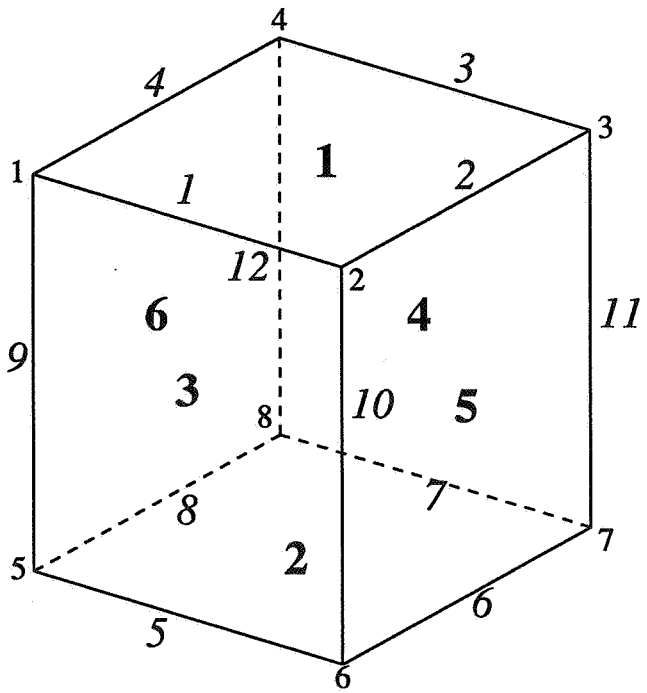


FIGURE 6. A cube

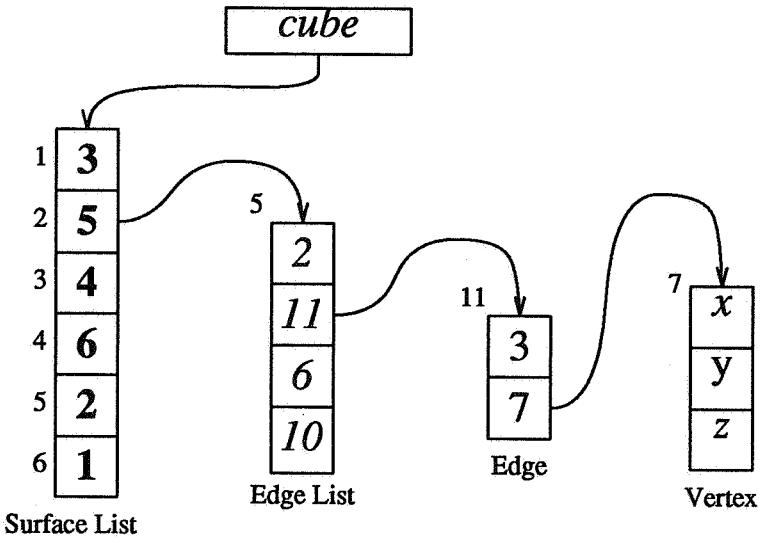


FIGURE 7. Simple hierarchical example of data structure of a cube

points and its direction. This will be implemented using pointers and lists.

The pointer-list structure can be used and typically implemented by the record data structure as in Pascal, such as:

```
type edge    = record
                   Id: integer;
                   StartPoint: ^point;
                   EndPoint: ^point;
                   PreviousEdge: ^edge;
                   NextEdge: ^edge
               end;
```

Instead of using the pointer-list structure, we may use a relational data model [12]. An example of surface information is shown in TABLE 2. This implementation requests ordering of items; otherwise, there is no concept of `PreviousEdge` or `NextEdge`. These two types of implementation imply typing in advance which results in a rigid data structure. They can be categorized as *intensional descriptions* mentioned in SECTION 2.2.2 for the following reasons:

TABLE 2. Relational data structure for surfaces (cf. FIGURE 6)

Surface	edge 1	edge 2	edge 3	edge 4
1	1	2	3	4
2	5	8	7	6
3	1	9	5	10
4	3	11	7	12
5	2	10	6	11
6	4	12	8	9

(1) An entity, such as an edge, has predefined abstract concepts. For instance, an **edge** has one attribute (**Id**) and four relationships (**StartPoint**, **EndPoint**, **PreviousEdge**, **NextEdge**) and these descriptions construct a hierarchy with a lower level (e.g., **point**).

(2) An edge structure, for instance, is a type which can be regarded as an element of a Cartesian product set. Therefore, once an edge structure is fixed with fixed number of attributes and relationships, it is very difficult to modify this rigid structure.

(3) This structure also represents constraints at the same time. For example,

```
StartPoint: ^point
```

indicates nothing but the constraint that the value of this field must be a pointer to a **point**.

(4) Thus, conventional data description methods are equivalent to the following *intensional* expression:

$$\begin{aligned} &cube\{(\mathbf{1}, \mathbf{2}, \cdots, \mathbf{6}, \mathit{1}, \mathit{2}, \cdots, \mathit{12}, 1, 2, \cdots, 8) \\ &\quad | \Sigma(\mathbf{1}, \mathbf{2}, \cdots, \mathbf{6}, \mathit{1}, \mathit{2}, \cdots, \mathit{12}, 1, 2, \cdots, 8)\}, \end{aligned}$$

where Σ implies the necessary conditions for this object to exist as a cube.

Note that we are insisting neither the pointer-list structure nor the relational database implementation is an intensional data description but the way we implement entities and its abstract concepts by them is intensional. It is even possible to implement an extensional data description using those types of data structure.

3.3.2. A Data Modeling Based on an Extensional Description Method

In an extensional description, predicates denote abstract concepts of an entity. The following facts represent the cube of FIGURE 6 in an extensional way. (Once again, notice that the use of predicate logic does not mean that this is the only way to describe things in an extensional way. These predicates might be implemented, for example, in a relational data model.)

vertex(1). ··· *vertex*(8).
line(*1*). ··· *line*(*12*).
surface(1). ··· *surface*(6).
Cube(*cube*).
has(*9*, 1). *has*(*9*, 5). ···
has(1, *1*). *has*(1, 2). ···
has(*cube*, 1). ··· *has*(*cube*, 6).

This representation has the advantage that it is easy to add new facts about entities. For instance, if this cube is a part of another complex object, the only thing we have to do is to add a new fact *has*(*complex-object*, *cube*), and we do not have to modify all other predicates. All the operations are realized by add operations.

This representation has disadvantages as well. We shall examine them more precisely in the next section.

3.3.3. Comparison of Extensional and Intensional Descriptions in CAD Applications

As pointed out in SECTION 2.4, intensional description methods which are used quite commonly in conventional CAD systems ignore slight differences in entity concepts and judge similar entities as different. We have seen in SECTION 3.2.2 that future CAD systems need a data description method which allows pattern matching. If we want pattern matching, it is fatal that we cannot precisely see slight differences and similarities of entity concepts.

In SECTION 3.1 the nature of CAD data operations was discussed. In particular, diversity and dynamic changeability are crucial in CAD applications. We need an integrated set of models each of which represents a different view of the design object, and might change during the design process. From this viewpoint, the facts that the data structure of intensional descriptions is rigid (see SECTION 2.4.1 and 3.3.1) and that data exchange between models may cause information loss and twist make intensional descriptions unsuitable for future CAD systems (see SECTION 2.4.2 and 2.4.3).

For the dynamic changeability of models, intensional descriptions are again not favorable. We have seen in SECTION 3.2 that in an extensional data description method most of CAD operations can be realized

by add operations which are less expensive than renewal operations. On the contrary, intensional descriptions make it difficult to modify the data schema.

Therefore, intensional descriptions must be avoided in future CAD systems and we should use extensional descriptions instead. However, we cannot simply replace intensional descriptions with extensional ones, because sometimes in practice intensional descriptions are than extensional ones.

Suppose we have the pointer-list data structure described in SECTION 3.3.1 and we must answer simple queries. For the question, *"What is the line next to line L0035?"*, the only thing we need to do is to trace the `NextEdge` pointer, which is reduced to just address computation. In an extensional description, we need explicit information about two adjoining edges, E_1 and E_2, like `nextEdge(E1, E2)`. If we do not have this information explicitly, we need to infer it using rules. In this case, clearly the intensional description is much faster.

Furthermore, extensional descriptions may sometimes lose computational semantics. For instance, in the example in SECTION 3.3.2, attributes of the cube, such as coordinates, edge length, etc., were not explicitly described. In case of the predicate logic representation we have to write three predicates which respectively correspond to the x, y, and z coordinates of a point. This apparently makes the performance of the system inefficient, as long as simple queries about coordinate values are concerned.

To sum up, although intensional data description methods have many disadvantages for the use in future CAD systems, they have also advantages in terms of implementation and efficiency. Therefore, we need to invent a new method somewhat in-between. This will be proposed in the next chapter.

4. A New Data Description Method

We have pointed out in SECTION 2.2 that an extensional description focuses only at the relationships among entities, while an intensional one focuses at the structure of entities. In SECTION 4.1, we shall try to combine these two methods into a new data description method. In SECTION 4.2, we consider an example of our method and show its power in solving many problems of conventional CAD systems as well as its ability to provide necessary functions for future systems.

4.1. Objects, Functions, and Predicates

An extensional description could be implemented naturally by a logic programming language such as Prolog [6]. On the other hand, an intensional description considers only entities and their attributes; even relationships are treated as if they were a kind of attributes (see the pointer-list implementation in FIGURE 7). It could be implemented by an object oriented language such as Smalltalk-80 [10]. From AI techniques, we may use MINSKY's frame theory [13].

Our goal is to integrate these two programming paradigms in the most natural way. There are a couple of results aimed at this goal [2, 14]. FIGURE 8 shows our solution which combines the object oriented programming paradigm with the logic programming paradigm. Small circles in this figure correspond to crosses in FIGURE 1 (a) which indicate entities, and an eclipse to topology which indicates abstract concepts of entities.

From now on, we call an entity an *object*. An object may have internal memory to store information just like slots in MINSKY's frame theory. But the internal structure of an object does not really matter, because we are interested not in how to store the information but in how to use it. In order to represent relationships among entities, we introduce *predicates*. This idea is similar to the *entity-relationship model* in database theory [5], but there is quite a difference in that in our paradigm relationships can be created, modified, and deleted all the time during the execution. In conventional database systems once the data schema is fixed, it will never be changed.

For instance, a fact that an object A is an automobile is denoted by *automobile*(A). We can naturally introduce the concept of class into our syntax; for example, the fact that all the automobiles are vehicles can be denoted by *vehicle*(X) :– *automobile*(X), where ":–" has the same meaning as in Prolog.

Access to the internal information of objects is done by invoking a *function*, such as *function*(*object*) → *value* for inquiry, or *function*(*object*) ← (any procedural definition) for definition. We can even define a function about relationships such as a function to compute the distance between two points,

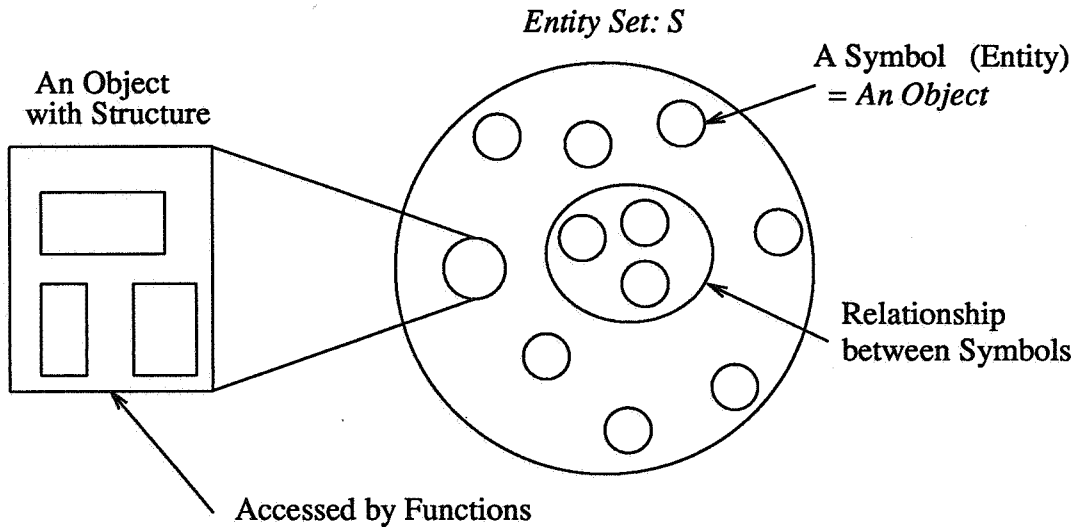


FIGURE 8. Objects and relationships

$$distance(p_1, p_2) \leftarrow sqrt((x\text{-}coord(p_1) - x\text{-}coord(p_2))^{**} + (y\text{-}coord(p_1) - y\text{-}coord(p_2))^{**} + (z\text{-}coord(p_1) - z\text{-}coord(p_2))^{**}),$$

where $x-$, $y-$, and $z-coord(p)$ are functions to get the x-, y-, and z-coordinate of a point p, respectively, *sqrt* is a function of square root, and x^{**} indicates the square of x.

We can generate a complex clause by combining simple clauses. Between the object and the predicate world, we have functions, and we can define complex functions by combining simple functions. In this context, we may have the following distinctions:

(1) A *function* is an intensional description of an entity and of a relationship between entities. It will be defined procedurally and reduced to primitive functions (built-in functions) so that they are finally executable.

(2) A function, $f_i(o)$, for an object, o, is expressing the value of an attribute f_i of o. It is also possible to express constraints among items, such as $f_1(o) \equiv f_2(o) + f_3(o)/2$.

(3) A *predicate* is an extensional description of objects. It will be defined declaratively by using primitive predicates.

(4) There is a connection between predicates and functions. For example, a predicate *greater-than* can be defined by

$$greater\text{-}than(x, y) \equiv \text{if } val(x) > val(y) \text{ then true else false,}$$

using a function *val* that returns the numerical value of attributes of objects.

4.2. Example

4.2.1. Effectiveness of Extensional Description of the Example

Let us consider a machine part which looks like a triangle. FIGURE 9 (a) is its two-dimensional representation. In FIGURE 9 (b), one of its corners is chamfered, while in FIGURE 9 (c) it is rounded. From a mechanical engineer's viewpoint, FIGURE 9 (a) is a rough sketch of this part and that FIGURE 9 (b) and (c) are more detailed drawings for manufacturing. This means that these three are basically identical and that, if

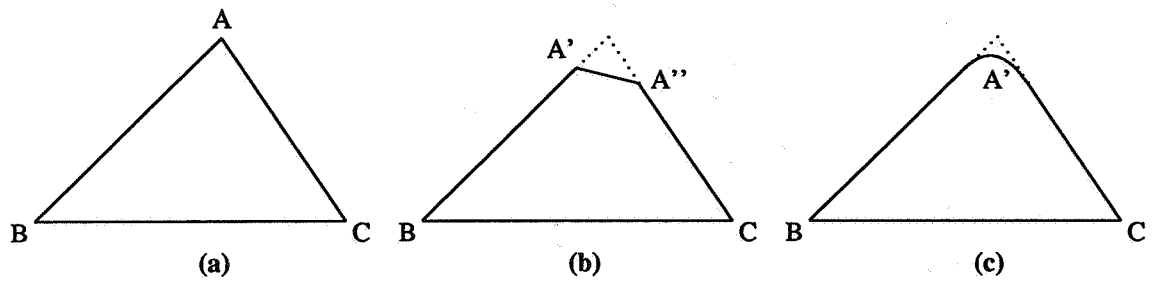


FIGURE 9. Three similar figures

something is changed in FIGURE 9 (a), this change should be propagated to other two properly.

We use a Prolog-like language; however, there is a difference. Let => be a symbol for an assertion operation; i.e., if the left hand side condition holds, the clauses on the right hand side will be asserted. Therefore, a rule

```
p => q(a), r(X)
```

is interpreted as:

> *If* `p` *is true, then* `q(a)` *and* `r(X)` *must be true. If either* `q(a)` *or* `r(X)` *is not known, it is added to the database.*

At the moment of assertion, objects might be created as side effect. If `q(a)` did not exist before the assertion, not only clause `q(a)` but also object `a` would be created and added to the database. When `r(X)` is asserted, an object would be created with the name `X` (or with a name given by the system), if it did not exist previously.

A polygon `POLG` is generated by the following rules (cf. [1]). (In the following pseudo program, `POLG` means a polygon, strings beginning with `L` and `l` indicate lines, and `P` and `p` indicate points.)

```
polygon(POLG, N) =>
      startpoint(P), endpoint(L, P, P1),
      M is N - 1, create(POLG, L, M).
create(POLG, L, 0) =>
      line(L), startpoint(P2), endpoint(L, P1, P2),
      has(POLG, L), has(POLG, P1), has(POLG, P2).
create(POLG, L, N) =>
      line(L), endpoint(L, P1, P2),
      has(POLG, L), has(POLG, P1), has(POLG, P2),
      M is N - 1, endpoint(L1, P2, P3), different(L, L1),
      create(POLG, L1, M).
line(L) =>
      has(L, P1), has(L, P2),
      point(P1), point(P2), different(P1, P2),
      endpoint(L, P1, P2).
```

The fact `startpoint(P)` is used to mark one of the vertices of a polygon, so that we can draw edges. The predicate, `endpoint(L, P1, P2)`, reads the line `L` has two end points, `P1` and `P2`. The predicate, `has(A, B)`, is representing so-called a part-assembly relationship that `A` owns `B`. The predicate, `different(X, Y)`, is true when `X` and `Y` are referring to different objects; otherwise, false.

A triangle `t` is created by using the rule;

```
triangle(T) =>
      polygon(T, 3).
```

and an assertion;

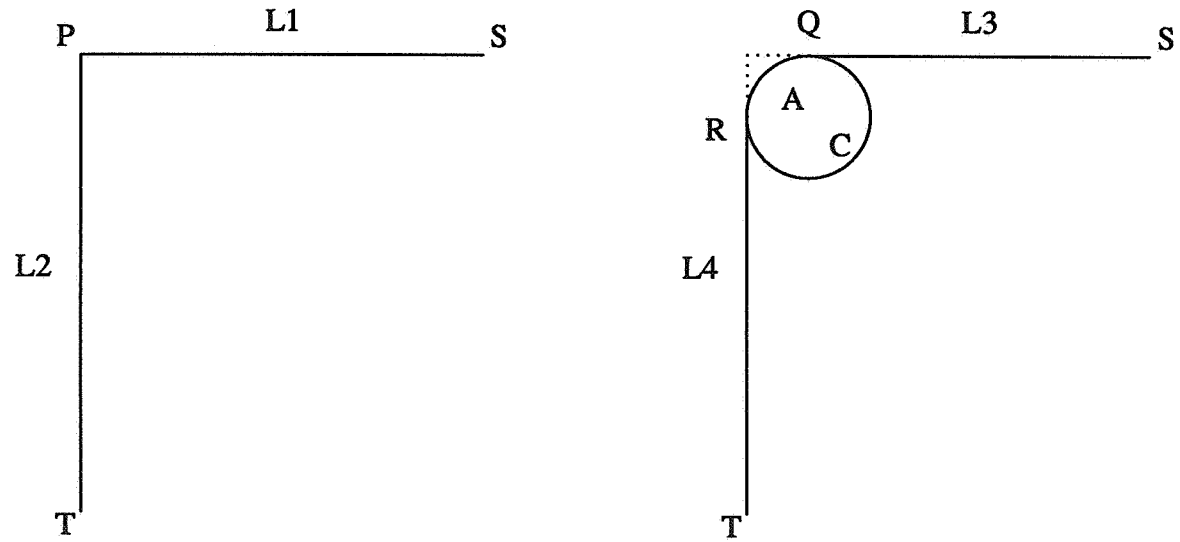
```
?- triangle(t).
```

Consequently, the following facts will be asserted and added to the database.

```
startpoint(p1).
has(t, l1).  has(t, l2).  has(t, l3).
has(t, p1).  has(t, p2).  has(t, p3).
has(l1, p1).  has(l1, p2).
has(l2, p2).  has(l2, p3).
has(l3, p3).  has(l3, p1).
point(p1).  point(p2).  point(p3).
endpoint(l1, p1, p2).  endpoint(l2, p2, p3).
endpoint(l3, p3, p1).
```

Now, consider rounding corner p2. (Chamfering is described in the same way.) First, we need knowledge for rounding a corner (FIGURE 10).

```
round(P, L1, L2, A, Q, R, L3, L4) =>
      point(P), line(L1), line(L2), point(Q), point(R),
      on(Q, L1), different(P, Q), on(R, L2), different(P, R),
      tangent(Q, C, L1), tangent(R, C, L2), circle(C),
      endpoint(A, Q, R), arc(A), has(C, A),
      has(A, Q), has(A, R),
      line(L3), endpoint(L1, S, P),
      endpoint(L3, S, Q), tangent(Q, C, L3),
      line(L4), endpoint(L2, P, T),
      endpoint(L4, T, R), tangent(R, C, L4).
      convex-arc(A, L3, L4).
```



(a) Before Rounding (b) After Rounding

FIGURE 10. Rounding a corner

The predicate, `on(P, L)`, holds, when a point `P` is on a (definite) line `L`. The predicate, `tangent(P, C, L)`, holds, when a line `L` is tangent to a circle `C` at a point `P`. The predicate, `convex-arc(A, L3, L4)`, holds, when an arc `A` form a convex arc corner of `L3` and `L4`.

Let `add` and `remove` explicitly add and remove objects to and from the database. Then, we can round a corner by the following query.

```
?- round(p2, l1, l2, A, R, Q, L1, L2),
   remove(p2), remove(l1), remove(l2),
   add(A), add(R), add(Q), add(L1), add(L2).
```

After this operation, we have the following facts in the database.

```
tangent(q, cir, l1).  tangent(r, cir, l2).
tangent(q, cir, m1).  tangent(r, cir, m2).
circle(cir).  has(cir, a).  convex(a, l3, l4).  arc(a).
has(m1, p1).  has(m1, q).  has(m2, p3).  has(m2, r).
endpoint(m1, p1, q).  endpoint(m2, p3, r).  endpoint(a, q, r).
has(a, q).  has(a, r).
point(q).  point(r).
on(q, l1).  on(r, l2).
startpoint(p1).
has(t, l3).  has(t, p1).  has(t, p3).  has(l3, p3).  has(l3, p1).
point(p1).  point(p3).  endpoint(l3, p3, p1).
```

Comparing the state of the database before the rounding and after, we find out the following.

(1) We have obtained complete separation of the states of the database before the rounding and after. This was achieved by the use of an extensional description; i.e., modification operations were realized by simple addition and removal of predicates which are cheaper than modifying them.

(2) Before the rounding operation we had the central model M (i.e., the first state of the database), and after the operation it became M_r (i.e., the second state of the database). The transition from M to M_r was carried out by the rounding operation, R; $M_r = R(M)$.

(3) Notice that this last operation is executed by add operations rather than renewal operations. M and M_r are separately established in different worlds. This means we have complete separation of models as well as inheritance of attributes and their changes. Therefore, if we regard M as the central model which is necessary for the integration of CAD systems, we can easily obtain different, independent models by having a function to derive them from M.

In SECTION 3.1, we have pointed out characteristics of CAD applications; i.e., diversity and dynamic changeability of the models, bulkiness of data, integrity and consistency control, time-spanning transactions, and multimedia. The first requirement, diversity and dynamic changeability of the models, can be satisfied by using an extensional data description. An extensional data description may also solve the problems of integrity and consistency control, because changes in one model can easily propagate to other models. The complete separation of models will solve the version control and the multiuser access control problems. (We admit that the problem of bulkiness and multimedia cannot be solved here.)

4.2.2. Introduction of Intensional Description Aspects

The predicate `on(P, L)` used in the previous example was not defined. It is not easy, if we can use only predicates. But, if we have data like the point positions and the line equations, the computation itself is by no means difficult. This problem is regarded as the problem how to integrate extensional information that predicates have and intensional information that may reside inside the objects. Let us remind that it is dangerous to simply mix up the object world and the predicate world, because there may be a distortion of information (SECTION 2.4). Thus, we have a contradictory requirement; we need to separate them, but still we need to integrate them.

A solution might be to use functions to bridge between the object and the predicate worlds (SECTION 4.1). Let `%f(X)` be a function to get information about an item `f` from an object `X`. Therefore, a point `P` can be associated with three functions, `%x(P)`, `%y(P)`, and `%z(P)`, to obtain its position. Using these functions, we can implement the predicate `on` crudely as follows[1].

```
on(P, L) =>
        endpoint(L, P1, P2),
        A is (%x(P) - %x(P1))/(%x(P2) - %x(P1)),
        B is (%y(P) - %y(P1))/(%y(P2) - %y(P1)),
        C is (%z(P) - %z(P1))/(%z(P2) - %z(P1)),
        A = B, B = C.
```

As we have already pointed out in SECTION 4.1, the internal information structure of objects is less important than the fact that functions can act as a bridge between the object and the predicate worlds which must be kept separate. In our solution, there still exists a boundary between the predicate and the object worlds. This separation is effective for the performance.

5. Conclusion

In this paper, we have proposed a new data description method for future CAD applications. The following are principal results.

(1) To describe entities and their abstract concepts (such as attributes and relationships among entities), there are two possible ways; i.e., an extensional description where we have abstract concepts as topology of the entity concept set, and an intensional description where we have entity concept set as topology of the abstract concept set.

(2) Although, basically, these two description methods are identical, there are several differences. For example, an extensional description method provides somewhat a wholistic view. On the other hand, in an intensional description method entities are constructed from predetermined abstract concepts.

(3) We found out that extensional data descriptions are more suitable for CAD systems, because *difficult* data operations (e.g., modifications) can be done by easy (hence cheap) operations. However, conventional CAD systems are usually implemented based on intensional data descriptions. This is primarily due to good performance for *simple* data operations.

(4) Therefore, we need to invent a new data description method which integrate extensional and intensional views and which satisfies requirements for future CAD systems. Our solution is to have two separated world, the object world representing entities and the predicate world representing abstract concepts, and functions between them.

We are now developing a prototype of IIICAD [17]. As the knowledge representation language for IIICAD, we are developing a data description language called IDDL (Integrated Data Description Language) based on the idea depicted in this paper [18, 19].

Acknowledgements

We are very grateful to the members of Bart Veth IIICAD group, Centre for Mathematics and Computer Science, for their enthusiastic support in completing this paper. Especially, we would like to thank Varol Akman for his useful critical remarks on an earlier version of the paper.

Reference

1. F. Arbab and J. M. Wing, "Geometric Reasoning: A New Paradigm for Processing Geometric Information," in *Design Theory for CAD, Proceedings of the IFIP W.G. 5.2 Working Conference 1985 (Tokyo)*, H. Yoshikawa and E. A. Warman (eds.), North-Holland, Amsterdam, (1987), pp. 145-165.
2. A. Bijl, "An Approach to Design Theory," in *Design Theory for CAD, Proceedings of the IFIP W.G. 5.2 Working Conference 1985 (Tokyo)*, H. Yoshikawa and E. A. Warman (eds.), North-Holland, Amsterdam, (1987), pp. 3-31.
3. D. G. Bobrow (ed.), *Qualitative Reasoning about Physical Systems*, North-Holland, Amsterdam, (1984).
4. D. G. Bobrow, S. Mittal and M. J. Stefik, "Expert Systems: Perils and Promise," *Communications of ACM*, 29(9), September 1986, pp. 880-894.

5. P. P. Chen, "The Entity-Relationship Model - Toward a Unified View of Data," *ACM Transactions on Database Systems*, 1(1), March 1976, pp. 9-36.

6. W. F. Clocksin and C. S. Mellish, *Programming in Prolog*, Springer, Berlin, Heidelberg, New York, (1981).

7. J. Encarnacao and F. Krause (eds.), *File Structures and Data Bases for CAD, Proceedings of the IFIP WG5.2 Working Conference 1981 (Seeheim)*, North-Holland, Amsterdam, (1982).

8. J. S. Gero (ed.), *Knowledge Engineering in Computer-Aided Design, Proceedings of the IFIP WG5.2 Working Conference 1984 (Budapest)*, North-Holland, Amsterdam, (1985).

9. J. S. Gero (ed.), *Expert Systems in Computer-Aided Design, Proceedings of the IFIP WG5.2 Working Conference 1987 (Sydney)*, North-Holland, Amsterdam, (1987).

10. A. Goldberg and D. Robson, *Smalltalk-80: The Language and its Implementation*, Addison Wesley, Reading, Mass., (1983).

11. S. G. Leahey (ed.), *Proceedings of COMPINT 85 (Computer Aided Technologies) (Montreal)*, IEEE Computer Society Press, Montreal, Quebec, Canada, (September 1985).

12. R. A. Lorie, "Issues in Database for Design Applications," in *File Structures and Data Bases for CAD, Proceedings of the IFIP WG5.2 Working Conference 1981 (Seeheim)*, J. Encarnacao and F. Krause (eds.), North-Holland, Amsterdam, (1982), pp. 213-222.

13. M. Minsky, "A Framework for Representing Knowledge," in *The Psychology of Computer Vision*, P. H. Winston (ed.), McGraw-Hill, New York, (1975), pp. 285.

14. E. Shapiro and A. Takeuchi, "Object Oriented Programming in Concurrent Prolog," *New Generation Computing*, (1983), pp. 25.

15. T. Tomiyama and H. Yoshikawa, "Requirements and Principles for Intelligent CAD Systems," in *Knowledge Engineering in Computer-Aided Design, Proceedings of the IFIP W.G. 5.2 Working Conference 1984 (Budapest)*, J. S. Gero (ed.), North-Holland, Amsterdam, (1985), pp. 1-23.

16. T. Tomiyama and H. Yoshikawa, "Extended General Design Theory," in *Design Theory for CAD, Proceedings of the IFIP W.G. 5.2 Working Conference 1985 (Tokyo)*, H. Yoshikawa and E. A. Warman (eds.), North-Holland, Amsterdam, (1987), pp. 95-130.

17. 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, (April 1987).

18. B. Veth, "An Integrated Data Description Language for Coding Design Knowledge," in *Intelligent CAD Systems 1: Theoretical and Methodological Aspects*, P. J. W. ten Hagen and T. Tomiyama (eds.), Springer, (in preparation, 1987).

19. B. Veth, "The Specifications of Integrated Data Description Language," CWI Report, Centre for Mathematics and Computer Science, Amsterdam, (in preparation, 1987).

20. H. Yoshikawa and E. A. Warman (eds.), *Design Theory for CAD Proceedings of the IFIP WG5.2 Working Conference 1985 (Tokyo)*, North-Holland, Amsterdam, (1987).