Database Model for Design Data

Iztok Savnik, Tomaž Mohorič*, Tomi Dolenc, Franc Novak

Department of Computer Science, Jožef Stefan Institute,
Jamova 39, 61111 Ljubljana, Slovenija,

*Faculty of Electrical Engineering and Computer Science,
University of Ljubljana, 61111 Ljubljana, Slovenija

Abstract

The paper presents a Construction Database Model *(CDM) used for the conceptual level modelling of design objects. A primary intention of the CDM model is modelling hierarchically structured objects and construction dependencies among objects. CDM is an attribute based data model. The CDM model represents various abstractions identified in the semantic database models GSM, IFO, SAM* and FDM by means of attributes. To distinguish between different abstractions, the notion of* attribute role *is introduced. Special attention is paid to the representation of derived properties of the object. They are described by means of a* constructor, *that can be compared to derived attributes as defined in [10, 4, 22, 8]. The constructor enables the description and reevaluation of the design object construction process. The CDM is used as a data model for the storage system Cons-Base, designed for storing data in CAD applications.*

1 Introduction

Design data encompasses data for describing entities modelled by the Computer Aided Design applications. Our work is concerned in particular with the designing of mechanical parts, their geometric representation, related technological data and tools for mechanical parts manufacturing. Hence, most examples in this paper will be drawn from geometric representations of mechanical parts.

Conventional data models are appropriate for representing large amount of the rigidly structured data usually stored in business applications, but they are not suitable for representation and manipulation of the design data. They do not provide constructs for representing hierarchically structured data, nor do they provide constructs for derived data definition and manipulation. For this purpose, database models Cactis [10], Trait [4], IFO [2], GSM [11], FDM [22], SAM* [24], $\boldsymbol{NF}^2$ model [18], IQL [3] and 3DIS [9] are more suitable due to their capability of representing structural and behavioural information relating to the design object. These models are appropriate for describing entities in the software engineering environment [16], computer integrated manufacturing [24] and computer aided design [15, 18, 9].

In the paper the, *Construction database model* (CDM) is presented. The CDM is intended for conceptual level modelling and manipulation of design data. The primary focus of CDM is in the modelling of the design-object hierarchical structure and the construction dependencies between objects. CDM is intended for modelling data in the design and engineering application environment. It can be implemented as an extension of the commmercial database system or can be used as a data model integrated in the target application environment. The paper describes the *structural part* of the CDM data model, whereas the data definition and manipulation language interface is not the subject of this work. The language will follow the ideas of the languages DAPLEX [22], $\boldsymbol{O^2FQL}$ [14], IQL [3] and FDL [17].

The CDM is an attribute based data model, therefore all object properties are described by attributes. The leading idea of the CDM model is representation of the various abstractions identified in the semantic database models GSM, IFO, $\boldsymbol{NF}^2$, SAM* and FDM with attributes. To distinguish between different abstractions, the notion of *attribute role* is introduced. Attribute role divides attributes into different categories on the basis of the attribute meaning. Attributes can be of ordinary, structural, derived of implicit role, representing ordinary properties, hierarchical structure, derived properties and implicit properties of the objects. Special attention is paid to the representation of the derived object properties described by a kind of derived attribute called *constructor*. The constructor enables the description and reevaluation of the design object construction process.

The CDM constructs are related to a number of recent data models. The definition of the attribute and its properties are taken from the functional data model FDM [22] and the Generic Semantic Model [11]. Mapping defined by the derived attribute is represented by the set of objects similar to the 3DIS model [9] definition of *mappings*. The idea for representing different abstractions by different kinds of types and attributes can be compared to the *associations* of the SAM* [24] data model, exept that the CDM enables the definition of different abstractions in terms of one type. The structural object hierarchy defined by the attributes of structural role can be compared with the complex object model [12] and the nested relations model as defined in [18]. The generalization abstraction is similar to the generalization abstraction as defined in the IFO model [2]. The idea of representing derived information by an attribute-like construct is taken from practical experience with geometric design, but can be compared with the derived attributes in the Cactis [10], Trait [4], FDM [22] and SDM [8].

CDM constructs are described in the following sections. First the notion of object and type is introduced. In Section 3 the basic types used for representing basic and atomic objects are described. Attributes used for representing relationships among objects are described in Section 4. The structured type used for organizing hierarchically structured objects is introduced in Section 5. Generalization is presented in Section 6. The construct used for representing construction dependencies among objects is also described. Finally, some implementation issues of the CDM prototype used for modelling and storing geometric design data are considered.

2 Basic concepts

Mechanical parts, tools related to the design process, and all entities used for describing mechanical parts are in CDM represented by *objects*. Objects may represent, for example, geometric objects, technological data, mechanical parts, NC machine tools and documentation. Every object has an associated *object identifier* (OID) which is independent of the object properties. The *object state* is represented by a set of values which represent object properties.

Properties of a collection of similar objects are represented by *types*. A type can be identified by its name, that is unique among the names of the types that participate in the conceptual scheme of the modelled data. Objects of a specified type are called *instances* of the type. A type is defined by the set of attributes which describe the properties of the corresponding instances. The formal notion of attribute is discussed in Section 5.

Types can be divided in two groups. Types that do not define any particular abstraction are basic and atomic types. The second group of types are nonatomic types, which describe one or more predefined abstractions. These abstractions are: generalization , structural abstraction and construction abstraction. Note that one type can describe more than one predefined abstraction. For example, a type can describe hierarchical structure and derived properties of its instances.

A *Conceptual scheme* is a description of the database from the semantic point of view. It is built from the collection of database model constructs. The CDM conceptual scheme comprises a set of types that are interconnected with the attributes.

3 Basic type

The *basic types* represent objects that can be printed or represented with any output device. Basic types are, for example: string, integer, real, circle, point or text. The set of basic types can be extended by any data structure that is widely used in the design system.

Basic types are graphically represented by the oval. The use of basic types is demonstrated in Figure 1. Examples of basic types are `Real` and `Point`.

4 Atomic type

Atomic types are used to represent objects without any predefined abstraction. Atomic type can, of course, represent abstraction which is implicitly defined by the user, but does not describe the abstraction explicitly defined by any CDM construct. Therefore, atomic objects are not represented as an aggregation of components and are not defined as a generalization of other type objects. Atomic object properties are described by attributes. The range of attributes describing atomic type can be objects of any defined type. Atomic objects represent simple mechanical parts, tools, documents, etc.

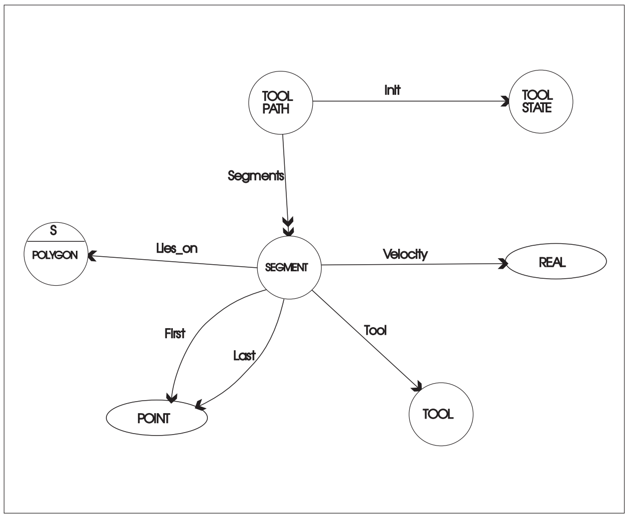

Figure 1: The description of the NC machine tool path

The symbol for describing an atomic type is a circle. The use of atomic type is demonstrated in Figure 1 where the NC machine tools path used for manufacturing a mechanical part is described. The path is composed of segments. Each segment includes information on tools movement velocity, the first and last point of the segment and the polygon on which the path is based. The examples of atomic types are `Tool_Path` `Path_Segment`, `Tool` and `Tool_state`.

5 Attributes

All object properties and relationships among objects are represented by means of *attributes*. In the formal definition of the attribute we will use the term *type domain*, which corresponds to all instances of that type. The domain of type T is written as $Dom(T)$.

Definition 1 *Let $D_1, D_2, \ldots, D_n, T$ be types. Attribute $\mathcal{A}$ is a mapping*

$$\mathcal{A} : Dom(D_1) \times Dom(D_2) \times \ldots Dom(D_n) \longrightarrow Dom(T),$$

which assigns to every ordered tuple $(d_1, \ldots, d_n)$, an object $t \in Dom(T)$ or a set of objects $\{t_1, t_2, \ldots, t_k\}$, where $t_i \in Dom(T)$.

N-tuples $(d_1, d_2, \ldots, d_n)$, where instances d_i are elements of the correspondent component type domain D_i, are *attribute arguments.* The set of all n-tuples, elements of $Dom(D_1) \times Dom(D_2) \times \ldots Dom(D_n)$, is called *attribute domain* and types $D_1, \ldots, D_n$ are called *domain types* of the attribute. In the case of binary attribute, attribute arguments are instances of a single type D and attribute domain is the set of all instances of type D. Objects t_i are called *attribute values.* The *attribute range* is the domain of the type T and the type T is called *range type* of the attribute. The attribute is graphically represented by an arrow.

CDM attributes have the following characteristics:

- The attribute has a name that is unique within the type and its family (i.e. predecessor and ancestor types in the generalization hierarchy, described in the Section 6).

- Depending on the attribute cardinality, attributes can have one or more values. We call them *singlevalued* or *multivalued* attributes, respectively. The value of the singlevalued attribute is single object and the value of the multivalued attribute is the set of objects from the attribute range. The singlevalued attribute is represented by a single arrow, while multivalued attribute is represented by a double arrow. The additional constraint specifying the cardinality of the multivalued attribute can be attached to the attribute definition. The constraint is defined by a *condition* of the form $(|A|\, e_i\, N)$, where A is an attribute, N is number and e_i is one of the comparison operators from the set $\{<, \leq, =, >, \geq\}$. The constraint is written under the arrow representing the attribute.

- The set of objects that are returned with an application of multivalued attribute can be ordered. The order can be defined when inserting the object in the set by the user or on the basis of the value of some object attribute. For example, points that constitute a polygon are ordered by the user. Points are sorted as they are connected in the polygon.

- The attribute role is a generalization of the "role" as defined in the Entity-Relationship model. Attribute roles divide attributes into different categories on the basis of the attribute meaning. An *ordinary attribute* (or just attribute) describes the object property. A *structural attribute* defines the relationships between hierarchically structured objects and their components. A *constructor* is a derived attribute by which the process of object construction is described. The constructor provides a description of the object derived properties. *Implicit attribute* describes the object property, which is implicit in the abstraction which is defined by a certain object type.

The use of attributes is demonstrated in Figure 1. The attribute `Velocity` is an example of a singlevalued attribute, and the attribute `Segments` is a multivalued attribute. All attributes presented in Figure 1 are ordinary. Attributes with other roles are presented in the following sections.

6 Generalization

When representing the design data, the need for organizing objects of different types in the collection frequently appears. For instance, we want to represent the set of objects which constitute polygon (Fig. 2). The set is composed of lines and arcs. This kind of the set object can be represented in the CDM by defining first the generalization type or the *supertype*. The supertype is defined by the set of *subtypes*, which are in this partiular case the types `Arc` and `Line`. The instances of all subtypes together make up the instances of the supertype. The set of arcs and lines can be than represented by a multivalued attribute which range are instances of supertype.

Each instance of a subtype is also an instance of the generalization type and opposite, each instance of supertype is an instance of the exactly one subtype. Thus, the sets of instances of subtypes are mutually exclusive. This constraint is usually accomplished in the design data environment. The formal definition of the supertype is the following.

Definition 2 *Let $T_1, T_2, \ldots, T_n$ be types. Generalization of types $T_1, T_2, \ldots, T_n$ is* **supertype** T*, if the following holds*

$$t \in Dom(T) \Rightarrow \exists! T_i : (t \in Dom(T_i) \wedge 1 \leq i \leq n)$$

The relationship between supertype and subtype is called the *ISA relationship*. The relationship defines every subtype instance as being also an instance of the supertype. The ISA relationship is represented by a double line ending with an arrow. The ISA relationships and correspondent subtypes and supertypes form a directed graph, called *generalization scheme*. The *family* of the type T consists of type T and its predecessor and ancestor types in the generalization scheme.

Let be $\mathcal{T} = \{T_i\}$ the set of subtypes T_i that are used for the definition of the particular supertype. Instances of subtypes T_i share some common properties, that are represented by the supertype attributes. Properties that are specific for the subtype instances are represented by attributes of the corresponding subtypes. Consequently, an instance of a specified type T includes the properties that are defined by the set of attributes $\mathcal{A}$, whose domain types are members of the family of the type T. To differentiate among the elements of the set $\mathcal{A}$, each attribute must have a unique name.

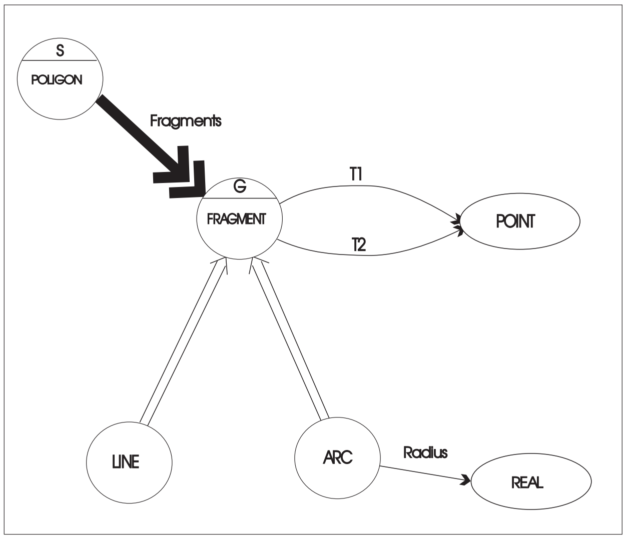

Figure 2: Polygon representation with the structural type and generalization.

Supertype is graphically represented with a circle labeled with a character G. The generalization is illustrated in Figure 2. The type *Fragment* is defined as the generalization of types `Arc` and `Line segment`. The first and the last fragment point is defined as the common property of both subtype objects. Instances of the type `Arc` include the attribute that describes the arc radius.

7 Structural type

In the CDM, the aggregation hierarchy of the complex design object is represented by *structural types*, which are described by attributes which have a structural role. These attributes are called *structural attributes*. Structural attributes are binary attributes that define the mapping between the object and its component objects. In the case of a multivalued structural attribute, the components are sets of objects, and if the structural attribute is singlevalued, the component is represented by one object. The domain of a structural attribute is a structural type domain and the range of a structural attribute is a component type domain (Fig. 3).

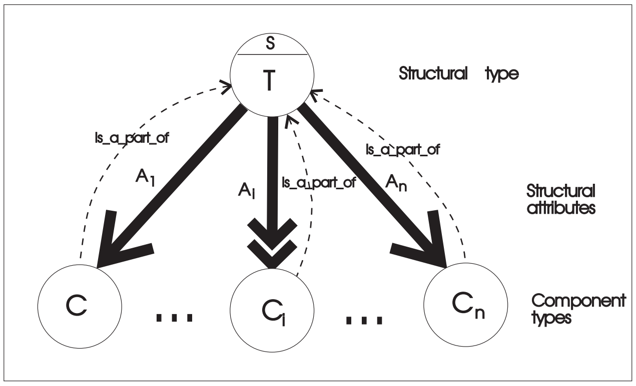

Figure 3: Structural type

Definition 3 *Type T is a* **structural** *type, if it serves as the domain type of the set of structural attributes $\{A_1, A_2, \ldots, A_n\}$, where $n \geq 1$.*

The definition of the structural attribute implies the definition of its inverse attribute, which is defined as an implicit attribute `Is_a_part_of`: its domain are component type instances and its range are structural type instances. `Is_a_part_of` attribute specifies that the component type object is a part of a more complex structural object. Therefore, implicite attribute enables the browsing of the structural object representation from simple to more complex components. In general, an object can be a part of more than one structural object, hence `Is_a_part_of` attribute is defined as multivalued.

The set of structural types together with the correspondent structural attributes form the acyclic graph called *structural scheme*. Vertices of the graph are structural types and arcs are represented by structural attributes. The structural scheme can be compared with *complex object type* [12] or with the nested relations [18, 1].

Additional properties of the structural objects (besides structural property) can be represented by the attributes of other roles. Note, that the structural type can be also defined as a supertype or a derived type. Hence, CDM enables the definition of more abstractions with the same type.

The use of structural type is illustrated in Figure 2. The polygon is composed of a set of fragments, which are organized in the order by which the fragments are physically interconnected. The component type `Fragment` includes the definition of the `Is_a_part` implicit attribute, which describes the fact that every fragment is a part of at least one polygon.

8 Representing derived objects

Object construction is the process of object generation and the calculation of object state from the states of the existing objects. The object which is calculated from the states of other objects is called the *derived object*. Object can be constructed by the set of provided *operations*. Instances of one type can be constructed by different operations.

Construction flow is an ordering of a set of objects and the corresponding construction operations, which can be presented as a directed acyclic graph. The vertices represent the objects and operations. The graph edges represent dependencies among the construction operations and the corresponding parameters and the resultant objects. Design systems typically store only objects that are the result of the construction operations. In this way the change of an object state requires a reconstruction of all objects that are influenced by the changed part of the design. Storing the complete design flow enables automatic reevaluation of those objects that have inconsistent state due to the changed object.

8.1 Constructor

The fact, that objects of a specified type can be constructed by an operation is described at the conceptual scheme level by means of a derived attribute called the *constructor*.

Definition 4 *Let be $T_1, T_2, \ldots, T_n$, and ,T types.* **Constructor** *$\mathcal{K}$ is defined by the mapping*

$$\mathcal{K} : Dom(T_1) \times Dom(T_2) \times \ldots \times Dom(T_n) \longrightarrow Dom(T)$$

and by the **construction operation** *which represents the construction dependency between the correlated objects.*

In other words, let t be an instance of the type T and t_i an instance of T_i. We may say that t is constructed (or derived) from the objects $t_1, t_2, \ldots, t_n$ if $t = \mathcal{K}(t_1, t_2, \ldots, t_n)$. As in the case of attributes, the objects $t_1, t_2, \ldots, t_n$ are called constructor *arguments* and object t the *constructor value*. Similarly, types T_i are the domain types and the type T is the range type of the constructor.

Figure 4a shows a typical constructor, which has two arguments. The constructor `Tangent` represents the construction of tangent line from a given point and a circle. The construction operation invoked by applying the constructor `Tangent` represents the construction dependency between the objects representing tangent line, point and circle.

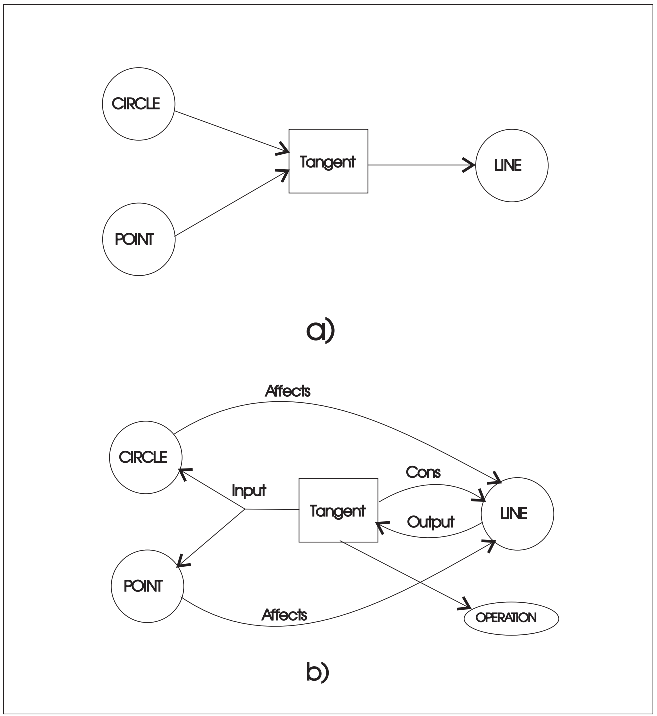

Figure 4: a) Constructor `Tangent` b) Constructor represented with type

Mapping defined by the constructor is represented by the *construction type* objects, called *operators*: these represent the link between the constructor value and the corresponding arguments and also identify construction operation. Therefore, the operator is an object which includes all necessary information for activating the construction operation on argument objects. In this way, the state of the object which represents the constructor value can be recomputed when needed. Constructor arguments, values and the operation are described by the corresponding construction type implicit attributes: `Arguments`, `Value` and `Operation`. The fact that the construction operation does not have solution is described with an implicite attribute `Solvable` (discused in the Section 9.2). The representation of construction type implicit attributes is described in Figure 4b.

In terms of the functional programming languages, constructor can be compared to the function that leaves the trail (operators) after its execution, so that the construction dependencies between objects can be reconstruced latter.

8.2 Derived type

Definition 5 *The type that serves as the range type of one or more constructors is a* derived type*.*

Although the derived type can be at the conceptual scheme level the range type of more than one constructors, a particular derived object can be constructed by an application of the exactly one constructor. Derived type is graphically represented by a circle labeled by character D.

The construction of each derived type object and its influence to the state of other objects is described by the set of implicit attributes (Fig.4b): `Affects`, `Cons` and `Changed`. The value of the attribute `Cons` is an operator identifying construction operation by which the derived object is constructed. The attribute `Affects` identifies all objects whose state depends on the state of its domain object. Finally, the value of the attribute `Changed` is a simple binary object, which describes the consistency of a derived object state.

When the state of a derived object is changed, the states of all objects that depend on the changed object become inconsistent. The object is directly dependent on some other objects when they are connected by an operator. Similarly, a object is indirectly dependent on other objects when they are connected with the chain of derived objects and operators. There are two basic operations for the maintenance of the state of the derived objects: *change propagation* and *reevaluation*.

The objects that are directly dependent on some object can be identified by the value of the attribute `Affects`. Inconsistency of the object state is marked by changing the attribute `Changed` to the value TRUE. When the state of an arbitrary object is changed, the state of all directly and indirectly dependent derived objects are marked as inconsistent.

The state of an inconsistent object is computed on the user demand. In this way, the object state can be changed many times before evaluating other dependent objects. As a consequence, elementary changes that may cause incorrect execution of the construction operations can be entered in the database. The evaluation of a derived object is executed when all necessary changes to the objects are completed.

9 Implementation

The Construction Database Model is used as a data model for the storage system Cons-Base, designed for storing data in CAD applications. In particular, the storage system is embedded in the system for the design of mechanical parts Cons-Cad. The system integrates geometric design of mechanical parts and their manufacturing by the numerically controlled machines. Cons-Cad and Cons-Base are implemented in Pascal for the IBM-PC computer.

Cons-Base is designed for storing information in a design environment. For efficient storing and manipulation of the design data, characteristics of the data were considered. Cons-Base has the following characteristics that are important for design object modelling and manipulation. Every object has an identification that is independent of its properties. This enables efficient implementation of the complex structural object. Basic geometric entities (e.g. circle,point,line etc.) are represented as basic objects and are therefore units of update and retrieve. Types representing metadata are stored in the same way as data objects. Recovery mechanism is implemented, and mechanisms for long duration transactions [13] are provided.

Cons-Base is composed of three layers. The low level layer of the Cons-Base is implemented as a modified virtual memory [21, 20] with shadow paging used for recovery mechanism. Virtual memory is upgraded by the segment layer, which is used for organizing the database into segments. The segment layer is primarily designed for efficient manipulation of average size segments that are grouped for storing data structures representing the design objects. In addition, large segments used for storing nonstandard design data such as bitmap images or texts, can be manipulated. Each design object is stored in one or more segments.

Basic data model constructs defined in the CDM are implemented in the Cons-Base *object layer*. The Cons-Base object layer includes routines for type definition and object manipulation. It enables the definition and manipulation of basic and atomic type objects only. The support for structural, generalization and derived type objects is implemented as an upgrade of the object layer.

9.1 Representing geometric constructions

In attemting to find the appropriate data model for geometric design, our aim was to represent the whole construction process in a way that allows local reevaluation of affected data after the change of a single construction parameter. This leads to the construction flow representation in terms of the constructors and corresponding implicit attributes as described in Section 8. The representation of geometric objects consists of two parts: numerical information (i.e. coordinates, parameters and equations), and

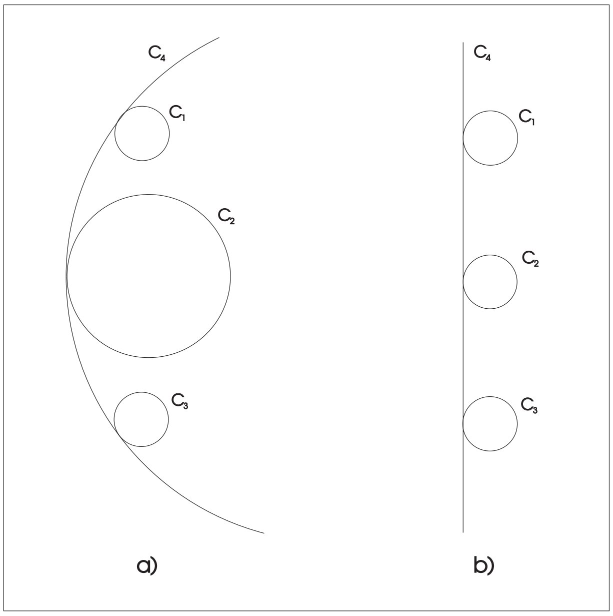

Figure 5: Tangential construction (a) and its degenerate case (b)

symbolic data specifiying relations among objects. In the geometric design, we want to represent explicitly the relations established by the geometric constructions. The symbolic links in the construction flow, realized as constructors, provide enough information to find construction-related geometric objects and corresponding geometric relations.

Dealing with the complex geometric constructions, especially when the automatic reevaluation is allowed after the parameter changes, there is an increasing possibility of singular configurations with degenerate solutions to occur. We may take as an example the situation in Fig. 5, where tangential construction of the circle C_4 is presented (a). The configuration becomes singular (b) due to the change of the radius $r2$ and the solution C_4 degenerates to the line. This may cause troubles to the automatic evaluation, because the algorithms for tangential lines calculation are different from those for circles. In the case that C_4 is a parameter to another geometric construction, the evaluation procedure would have to check many alternatives at each step. To avoid these problems as much as possible, an uniform representation for the elementary geometric objects in plane (points, lines, circles) was developed [5]. At the top operational level, these objects are treated as equal, and calculations run independently of the type of the objects involved. Therefore it is not distinguishable whether the resulting object is a point, a circle or a line but through the interpretation of its data. This generalization is easily realized in our database model. The elementary geometric objects in Cons-Base are accessed and modified via the supertype, and only the special data is separately maintained in the subtype attributes.

During the change propagation process, some of the geometric relations among objects may break and consequently some construction operations become unsolvable. For instance, if two circles that are the parameter objects of the construction operation "*find_intersection_points*" do not intersect, due to the change or their radius or center points, then the operation is unsolvable. As the unsolvable construction operations become apparent after the attempt of evaluation of changed objects, many gradual changes of the objects in the construction flow are allowed before the actual evaluation. In this way, the construction flow can be temporary inconsistent and even unsolvable. If the evaluation is activated in spite of unsolvable construction operations, which are caused by the mistake or a misconception of the design, the unsolvable constructors are marked by setting attributes `Solvable` to FALSE. This indicates that all objects which depend on unsolvable constructors are inconsistent with objects predecessing unsolvable constructions.

10 Conclusion

The structural part of the Construction Database Model used for representation of the structural and derived properties of the design data is presented. The CDM is based on the representation of modelled entities with objects and attributes. Different abstractions can be defined by attaching different roles to attributes. In this way, the structural object hierarchy and construction dependencies between design objects can be defined. The generalization object hierarchy can be defined by supertype definitions. A unique construct for describing dependencies between derived objects called "constructor", is introduced. The use of the constructor provides a tool for the description of object design process inside the database. In this way, computation which usually takes place in the application code environment is performed at the database system level.

The CDM can be used as a data model for modelling data in the CAD environment or in related engineering applications. It can be implemented as an extension of the commercial database system or can be used as a data model integrated in the target engineering application. Embedding the consistent data model for modelling design data in application environment improves its data organization and simplifies coding of application.

References

[1] Abiteboul S., Bidoit.N., Non First Normal Form Relations: An Algebra Allowing Data Restructuring, Jurnal of Comp. and System Science 33, 361-393, 1986

[2] Abiteboul, S., Hull, R., *IFO: A Formal Semantic Database Model*, ACM Trans. Database Syst. 12, 4 (1987), 525-565

[3] S. Abiteboul, P.C. Kanellakis, *Object Identity as Query Language Primitive*, ACM SIGMOD 1988

[4] Tzi-cker Chiueh, Randy Katz, *Trait: An Attribute Management System for VLSI Design Objects*, ACM SIGMOD, Volume20, Issue 2, June 1991

[5] Tomi Dolenc, *The model for geometric object representation in CAD with cyclographic projections*, Masters thessis (in Slovene), Ljubljana 1990

[6] Susi Dulli, Vitaliano Milanese, *A DataBase Design for Graphical Models*, ACM SIGPLAN Notices, Vol.25, No.12, Dec.1990

[7] Antonio L.Furtado, Erich J.Neuhold, *Formal Techniques For Data Base Design*, Springer-Verlag, 1986

[8] Hammer, M., McLeod, D., *Database Description with SDM: A Semantic Database Model*, ACM Trans. Database Syst. 6, 3 (1981), 351-386

[9] Hamideh Afsarmanesh, Dennis McLoad *The 3DIS: An Extensible Object-Oriented Information Management Environment*, ACM Trans. on Inf. Sys., Vol.7, No.4, Oct.1989

[10] Scott E.Hudson, Roger King, *Cactis: A Self-Adaptive, Concurrent Implementation of an Object-Oriented Database Management System*, ACM Trans. on Database Sys., Vol.14, No.3, September 1989

[11] Ritchard Hull, Roger King, *Semantic Database Modeling: Survey, Applications, and Research Issues*, ACM Computing Surveys, Vol.19, No.3, September 1987

[12] Richard Hull, *Four views of Complex Objects: A Sophisticate's approach*, Lecture Notes on Computer Science 361, Nested Relations and Complex Objects, edited by S. Abiteboul, P.C.Ficher, H.-J.Schek,

[13] Katz, R.H, *Computer-Aided Design Data*, 9. chapter in the *New Directions Database Systems*, Ariav, G., Cliford, J.,(Editors), Ablex Publishing Corporation, Nowwood, New Jersey, 1986,

[14] Michael V.Mannino, In Jun Choi, Don S.Batory The Object-Oriented Functional Data Language IEEE Trans. on Soft. Eng., Vol.16, No.11, Nov.1990

[15] S.B. Navathe, A. Cornelio, *Modellig Physical Systems by Complex Structural Objects and Complex Functional Objects*, Int. Conf. on Extending Database Technology 1990, published in Lecture notes in Computer Science 416

[16] M.P.Papazoglou, L.Marinos, *On Integrating Database Modelling Constructs for Software Engineering Databases*, Microprocessing and Microprogramming 27, 1989, 113-120

[17] Aleksandra Poulovassilis, Peter King Extending the Functional Data Model to Computational Completness Int. Conf. on Extending Database Technology 1990, published in Lecture notes in Computer Science 416

[18] M.A.Roth, H.F.Korth, A.Silberschatz *Extended algebra and calculus for non 1NF elational databases*, ACM Trans. Database System 1988, Vol.13, No.4, 389-417

[19] Toby J.Teorey, Dongqing Yang, James P.Fry, *A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model*, ACM Computing Surveys, Vol.18, No.2, June 1986

[20] Savnik Iztok, Novak Franc, *A Construction Database Model*, Microprocessing and Microprogramming 27 (1989), 121-126

[21] Iztok Savnik, *Semantic Database Model for Computer Aided Design*, Masters thesis (in Slovene), 1991

[22] Shipman, D. W., *The Functional Data Model and The Data Language DAPLEX*, ACM Trans. Database Syst. 6, 1, (1981), 140-173

[23] John Miles Smith, Diane C.P.Smith,*Database abstractions: Aggregation*, Comm. of the ACM, June 1977, Vol.20, No.6

[24] Stanley Y.Su, *Modelling Integrated Manufacturing Data with SAM**, IEEE Computer, January 1986