EXPERT DATABASE SUPPORT FOR CONSISTENT DYNAMIC OBJECTS

NGUYEN G.T , RIEU D.

INRIA & Universite de Grenoble
Laboratoire de Genie Informatique
BP 68
38402 St-MARTIN-D'HERES
France
nguyen@imag.UUCP

ABSTRACT

The advent of new database applications such as engineering design stresses the need for new functionalities in database systems. It includes the management of multiple representations for database objects, long transactions as well as dynamic data structures. This paper presents the approach used in **CADB**, a prototype expert database system dedicated to CAD, for the management and control of the consistency of design objects. It concerns both the operations on the object **property values** and the interactive manipulation of their **structure**.
They involve concepts of the object models as well as the application semantics. They rely therefore on concepts used for **representing** the design objects and on semantic notions related to **expert knowledge** in the application domain. Among these are the notions of **consistency** and of **completeness** of the objects.
These two complementary aspects are detailed and their relationships described. The emphasis is on the **heuristic rules** that provide a unified knowledge-based approach for their management independent of the particular application being considered. **Dynamic inheritance** mechanisms are also presented that support the manipulations performed on the object structures. It is shown how they help providing expert database facilities.
Key-words : databases, expert databases, knowledge bases, objects, inheritance, consistency, logic, heuristic rules, CAD.

This work is supported in part by the French Department of Telecommunications CNET and the French Department of Research, Program PRC-BD3.

1. INTRODUCTION

Integrated systems for Computer-Aided Design applications (CAD) usually support :
- the definition of a design process,
- expert knowledge encoded in ad-hoc repositories,
- various design tools such as graphic interfaces and specific application programs,
- a common, reliable and sharable database that is eventually connected to private or semi-public databases [8].

Current research in the field has given rise to specific proposals and few software implementations [2, 6, 9, 11]. Expert system technology has also enlightened new possibilities for assisting the designers in deriving information and controlling the consistency of their design [16]. Recent studies have proposed sophisticated techniques for integrity constraint enforcement in knowledge and data bases [7]. However, integrating available design tools with full-scale expert systems and generalized databases is still an open issue [17].
This paper presents some features intended to enhance **CADB**, a database system dedicated to CAD, into an Expert Database System. It extends previous work on knowledge-based integrity constraint validation [14] by providing semantic constraint enforcement on derived and dependent information. A prototype of **CADB** is currently implemented using a relational database system and a deductive component [4].
It builds on database and expert system research for the design of an integrated architecture implementing new functionalities for CAD, including :
- powerful modelling capabilities,
- sophisticated modelling assistance,
- the use and maintenance of expert knowledge, e.g design rules,
- the management of high-level artifacts, e.g logical VLSI designs.

The emphasis is here on the **automated interactions** between the domain specific knowledge and the application data. They are directly implemented as part of the system's capabilities. They rely on :
- a common **database** sharable by a community of designers working in parallel on the same design objects,
- **expert knowledge** encoded in a **deductive** knowledge base that is connected on-line to the database. It is used for controlling **automatically** the semantic integrity and for the calculation of the side-effects during the information updates.

These two components are used to detect **automatically** the side-effects of the modifications requested by the designers. They stand as a **watchdog**, i.e an early warning feature, to make the designers aware of the potential consequences of their actions. As such, they implement a **pessimistic** approach to semantic integrity control. This is studied here in a engineering design environment.

The architecture of **CADB** is briefly described in the remainder of this section (Figure 1).
CADB is currently implemented in Prolog on APOLLO^{TM} workstations. A prototype in OPS5 is also designed on a VAX^{TM} 11/785 running Unix^{TM} 4.3 BSD.
The underlying database is a general purpose relational database system developed at IMAG called **MICROBE** [10]. It is implemented in Pascal and C and includes over 15,000 lines of code. An extended version has been implemented by the CAD Research Department at CNET (French National Research Center for Telecommunications) for VLSI circuit design [5]. It includes multiple databases organized as a hierarchy of a common database and private working databases.
The knowledge base is implemented in Prolog. It includes over 16,000 clauses and is expected to go well over 25,000 when completed. The particular Prolog implementation used is a commercially available release allowing both-ways communication with external programs written in C or Pascal.
The two components are tightly connected together. The database acts as a **back-end** data repository for the knowledge base. It stores the multi-level object data : e.g logic, symbolic and layout representations for VLSI circuits as well as the various object instances [1].
The knowledge base stores the admissible operating rules concerning :

- the modifications of the object structures,
- the specific design rules at hand,
- and specific **heuristics** for the automatic consistency controls implemented.

The interface with existing CAD tools such as simulators and graphic layout editors is guaranteed through a specific component called the **logic interface** (LI) which is common to the database (for data extraction and manipulation) and to the expert knowledge component (for side-effect calculation and optimization). It includes :

- a **validation processor** (VP) which triggers the appropriate components upon interaction with the designers. It uses a meta-knowledge base and specific knowledge bases. The meta-knowledge base includes the rules specifying the admissible operations for the manipulation of the concepts in the object model, e.g the object stuctures. The application knowledge bases include the specific rules relevant to the design process under consideration, e.g C-MOS technology for VLSI circuits.
- a **designer interface** (DI) which implements the interface between **CADB** and the user. It is a high-level **object-oriented** interface for the definition and update of the design objects [15].

Section 2 introduces the notion of objects in **CADB**. Section 3 describes the automatic computation of the object **classes** in **CADB** based on the systematic invocation of **optimization rules**. Section 4 defines the operating rules implementing the **certification** of the update operations, i.e the control of their correctness depending on the **consistency** and the **completeness** of the objects. Section 5 is a conclusion.

2. OBJECTS IN CADB

A specific requirement of advanced applications like CAD concerns the object structures : they evolve over time in contrast with more traditional data management only requires static database schemas. Engineering design must therefore support dynamic objects in both their evolving property values and changing models [16].
CADB puts an emphasis on upward model designs. An object structure may therefore be augmented only with existing object models and constraints.
During their manipulation, object consistency and completeness are dynamically examined to provide their actual **state**, which can be one of the following :
- incomplete and consistent, meaning that the design improves,
- incomplete and inconsistent, meaning that it does not improve,
- complete and inconsistent meaning that the design is wrong,
- complete and consistent meaning that the design is correct.
Unfortunately, in most available database systems and engineering environments, object consistency can only be examined **a posteriori**, i.e when the updates have been performed.
A novel feature in **CADB** is that the designers can invoke a **preventive** control on the updates. The immediate impact of their actions on the design objects as well as the side-effects of the modifications can be computed automatically. This requires sophisticated mechanisms for propagating updates and deriving information. First-order logic and heuristic rules are used for this purpose. The rules are independent of any specific object model and are only concerned with the application semantics. Examples are given in Sections 3 and 4.
Another feature in **CADB** concerns the **optimization** of the update controls. The system stores the fact that a particular operation failed. It is therefore unnecessary to control it later for "equivalent" objects (in the sense defined in Section 2.2). The modelling concepts and the notions of object classes and certification are defined in the following sections (2.1 to 2.4). First-order logic is used for their formalization. In the following, the symbol ": -" is used for the logical implication and the symbol "&" for the logical conjunction.

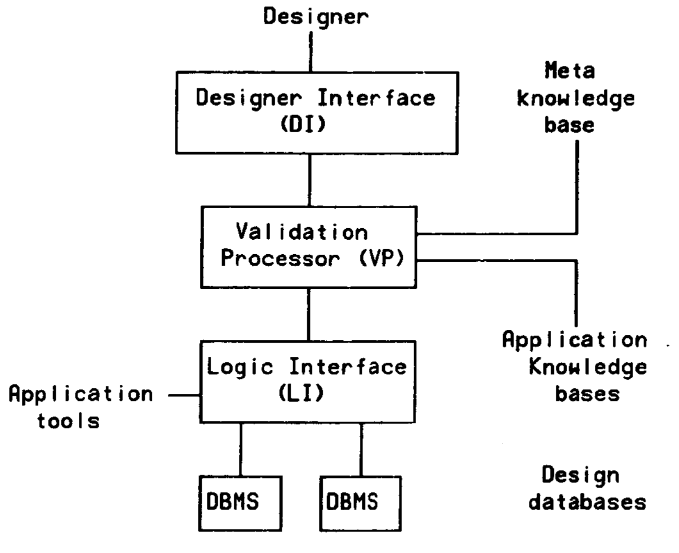

Figure 1. Architecture of **CADB**.

2.1 Modeling concepts

In the following, the term object **schema** is used for the term object **structure** in the usual database terminology. Object **schemas** are defined by :

- **specification rules** defining the object properties and structure,
- **derivation rules** called **links** specifying calculated properties,
- **integrity constraints** [12].

For example, the objects Point and Segment are defined as described in Figure 2.

```
DEF_E Point                 DEF_E Segment
| /* define point */        | /* define segment set */
| abs   :   Real            | org     : Point
| ord   :   Real            | ext     : Point
FDEF_E                      | lg      : Real
                            | not(conf(org,ext))    IC1
                            | lg := dist(org,ext)   L1
                            FDEF_E
```

Figure 2. Definition of Point and Segment.

A Point is defined by its properties abs and ord (x and y real coordinates respectively). A Segment is defined by its properties : origin and end points (org and ext resp.) and its length lg. They are the specification rules of the objects. The length lg is defined by the link L1, i.e the distance between org and ext points : `lg := dist(org,ext)` which acts as a derivation rule. An integrity constraint IC1 forces both org end ext points to be different.

These definitions are transformed internally in **CADB** into first-order clauses :

```
point(p)    :- name(p,n) & string(n) & abs(p,x)
               & real(x) & ord(p,y) & real(y)
segment(s)  :- name(s,n) & string(n) & org(s,o)
               & point(o) & ext(s,e) & point(e)
               & lg(s,l) & real(l)
               & not(conf(o,e))
               & equal(l,dist(o,e)).
```

2.2 Object classes

Object classes are defined in **CADB** by conjunctions of first-order atomic formulae belonging to object schemas, i.e specification rules, derivation rules and integrity constraints. The class C1 of all Points for which only the name and x-coordinate are known is defined by : `name(p,n) & string(p,n) & abs(p,x) & real(x)`. The definition of all such classes is obtained by computing the closure set of all the conjunctions of atomic formulae. A set of heuristic optimization rules that reduces this set of classes to the **relevant** classes only is described in Section 3.

> *Definition :* A class C1 is **smaller** than a class C2 if the conjunction of formulae defining C1 is a proper subset of the conjunction defining C2 : C1 < C2. For example the class C1 defined by : `name(p,n) & string(n)` is smaller than the class C2 defined by : `name(p,n) & string(n) & abs(p,x) & real(x)`. Class C2 is larger than class C1.

It should be clear that there may be several classes of (maybe incomplete) objects corresponding to the same object structure. In the example above, C1 and C2 are two classes of incomplete points corresponding to the definition given in Section 2.1.

> *Definition :* Two objects belonging to the same class are **equivalent**. The same conjunction of formulae hold for all the objects in a particular class (i.e they comply with the same set of constraints).

Incomplete objects are taken into account here since all potential and relevant classes are produced. An object can therefore be attached to exactly one **maximal** class at any time. Such a class is defined by the largest conjunction of satisfiable atomic formulae for the object. The class C1 above contains all points with unknown y-coordinate.

Inconsistent objects are also taken into account here because the negation of a constraint defines also an object class. In the following, a **decidable** constraint is a constraint that can be proved true or false using the actual state of the system, including the data items stored in the design databases. This is consistent with the notion of satisfiability of formulae in first-order logic. The equivalence classes of objects are extended here to take this into account. A decidably false constraint defines an object class in a similar way than decidably true constraints do.

A specific element called **prototype** is maintained for each class. It is produced automatically with the first object instance in the class and maintained dynamically to reflect the updates performed on the objects belonging to the class.

Prototypes model in **CADB** dynamic data structures and values. They form a database **abstract** where errors on the object structures are prohibited but property value errors are permitted.

2.3 Automatic consistency control

The idea used here is that considering the initial schema of an objet, it is possible to produce automatically all the potential structures of the sub-objects it may contain. **The only legal structures are those produced this way**. The incremental design of an object can therefore be controlled automatically by verifying that all the operations on a sub-object transfer **it** from a legal object class (i.e structure) to another legal class (i.e structure).

The paradigm used to control the update operations on the design objects consists in detecting the potential inconsistencies resulting from the user actions as soon as possible. The control is devided in two phases. The first one is the **certification** of the operations. The second is their **validation**.

The certification controls the operations by first applying them on the prototypes of the objects involved.

The validation makes the operations effective on the actual database items.

> *Definition* : An operation on the prototype of an object X belonging to class C1 is **certified** iff its application on its prototype produces an object X' belonging to a class C2 such that : C1 = C2 or C1 < C2.

If one of the two following cases occurs, an update operation is not certified :

- an undefined object or constraint has been referenced,
- a constraint is no longer satisfied by the object.

2.4 Characterizing object classes

We define in this section the paradigm for adapting the concept of certification to the design environment. It includes :

- **optimization rules** that correspond to the knowledge available in the application domain. They are used to reduce the number of object classes to only the **relevant** ones and thus optimize the **automatic** characterization of an object class when it is modified. This characterization will result in the acceptance or rejection of the operation. In the first case, the modified object belongs to a legal class. In the second case it belongs to an illegal class (Section 3).
- **operating rules** for the control of the designers operations. They allow a refinement of the order relation "<" defined in Section 2.2 between classes produced from a particular object schema. It will determine the validity of an update operation. An operation will be certified only if the object prototype involved is transfered to a larger class (Section 4).

These rules depend on the application semantics and on the object schemas. They are stored in the system's knowledge bases. They allow an automatic interaction between the Validation Processor and both the designers operations and the application data (Figure 1).
The **certification** of the users operations involves processing them on the prototypes for the objects involved. Characterizing an object class is therefore performed for every update operation. Depending on this is the correctness of the entire operation. It must therefore be very efficient. It depends on the number of classes to examine. The optimization rules mentionned above allow a significant reduction in their total number. Further, the result of an operation on a prototype is stored so that non certified operations are not tested later for equivalent objects.

2.5 Manipulating object structures

The need for dynamic data structures is widely recognized as a basic requirement for advanced database applications such as engineering design. **CADB** supports dynamic object schemas in order to modify existing object structures or define new objects. The creation of composite objects is also allowed. Elementary or predefined objects are supposed to be defined by composition of basic objects like integers, real, character strings, pixels and bit strings that are not subject to structure changes.
Inheritance of the object instances among classes requires here new mechanisms enforcing the correct propagation of the schema modifications on the object instances where appropriate.
The modifications are modeled as finite sequences of reduction, augmentation, connection and product operations. Intuitively, they model the deletion and addition of properties, links or constraints to an object schema and the composition of two or more existing schemas to define a composite object.
Let C1 and C2 be two object class definitions and F any conjunction of atomic first-order formulae.

The **reduction** of C1 by F (noted C1 - F) defines the objects satisfying the formulae defining C1 except those in F.
The **augmentation** of C1 with F (noted C1 + F) defines the objects satisfying all formulae in C1 and F.
The **connection** of C1 and C2 (noted C1 x C2) defines the objects satisfying both the formulae defining C1 and C2.
The **product** of C1 by C2 (noted C1 * C2) defines the objects satisfying the formulae involved in C1 and C2 and a specified condition C. It is defined only if the definitions of C1 and C2 are not disjoint.
Further details and examples are given in [13].
It seems predictable that the integration of these operations with the automatic consistency controls presented above will provide a powerful and flexible support for the manipulation of dynamic objects.
These aspects are detailed here for CAD applications. Simple examples show their usefuleness and practicability. Implementing this approach in other application areas seems possible but requires further investigations.

3. AUTOMATIC CHARACTERIZATION OF OBJECT CLASSES

A coarse computation of the closure set of all conjunctions of the atomic formulae belonging to an object schema would lead to a large number of irrelevant classes. They may be meaningless because inconsistent (e.g defining the length of a point) or because the objects are forced in the application to conform to specific configurations (e.g segments with at least abs and org points). This may be the case when the object models used do not prohibit specific inconsistencies (weak constraints allowing for example incomplete objects) or when they are in contrast used as strong integrity constraints (e.g no undefined properties).

Recall the example defined above for Point and Segment objects :

```
point(p)   :- name(p,n) & string(n) & abs(p,x)
              & real(x) & ord(p,y) & real(y)
segment(s) :- name(s,n) & string(n) & org(s,o)
              & point(o) & ext(s,e) & point(e)
              & lg(s,l) & real(l)
              & not(conf(o,e))
              & equal(l,dist(o,e)).
```

The length of a segment can here be calculated by `dist(o,e)`, i.e when the two points o and e are known. The class defined by : `name(s,n) & string(n) & lg(s,l) & real(l)` is therefore irrelevant here.

The following rules are used to characterize the **relevant** classes. They are implemented as part of the system standard features for consistency controls. Further, they extend the approach described in [14] by providing a systematic knowledge-based constraint enforcement paradigm for derived and inter-dependent object properties.

R1 - class definitions must include the type constraints corresponding to their properties :

CAD systems usually provide upward design methodologies. Thus, classes including a property definition and the negation of its particular type are **discarded** : a segment cannot belong to the class defined by :

```
segment(s) :- name(s,n) & string(n)
              & org(s,o) & not(point(o)).
```

Further, type checking for objects and their properties is systematically performed for each instantiation and modification. Relevant classes therefore include only those that have for every object or property the corresponding type : `segment(s) :- org(s,o)` is discarded because `point(o)` is missing. In contrast, `segment(s) :- org(s,o) & point(o)` is relevant.

R2 - class definitions must include the specification rules for the name of their instances :

Object occurences are given a unique identifier. The instantiation of `name(x,n)` is therefore necessary. Together with rule R1 above, this implies that : `name(x,n) & string(n)` is necessary for each object class. All other classes are discarded.

R3 - class definitions having derived properties must include the specification rules for all their arguments :

The length of a segment may be computed iff the org and ext points are known. This rules discards all the classes involving constraints or functions with missing arguments : `segment(s) :- name(s,n) & string(n) & not(conf(o,e))` is discarded.

R4 - class definitions must include all the decidable constraints corresponding to their properties :

CADB implements an immediate consistency control. If a constraint is decidable because its arguments are instantiated, it must be present in the class definition. The class defined by :

```
segment(s) :- name(s,n) & string(n)
              & org(s,o) & point(o)
              & ext(s,e) & point(e)
```

is therefore discarded. The org and ext point being instantiated, the constraint : `not(conf(o,e))` should be enforced.

All decidable constraints must be evaluated. In **CADB**, they correspond to weak consistency rules. The following definitions are therefore relevant :

```
segment(s) :- name(s,n) & string(n)
              & org(s,o) & point(o)
              & ext(s,e) & point (e)
              & not (conf(o,e))
              & lg(s,l) & real(l)
              & equal(l,dist(o,e))
segment(s) :- name(s,n) & string(n)
              & org(s,o) & point(o)
              & ext(s,e) & point(e)
              & conf(o,e) & lg(s,l)
              & real(l)
              & equal(l,dist(o,e))
```

R5 - class definitions must include all their decidable links :

CADB implements an automatic derivation of informations. Thus, if all the arguments of a link such as `dist(o,e)` are instantiated, it is automatically derived. The class defined by : `segment(s) :- name(s,n) & string(n) & org(s,o) & point(o) & ext(s,e) & point(e) & not(conf(o,e))` is discarded because although decidable the conjunction : `lg(s,l) & real(l) & equal(l,dist(o,e))` is missing.

R6 - class definitions must include only their defined links :

A link acts in **CADB** as a derivation rule. It corresponds therefore to a strong consistency rule. This eliminates the classes involving the negation of a link. The class defined by :

```
segment(s) :- name(s,n) & string(n)
              & org(s,o) & point(o)
              & ext(s,e) & point(e)
              & not (conf (o,e))
              & lg(s,l) & real(l)
              & not(equal(l,dist(o,e)))
```

is therefore discarded.

R7 - class definitions must include the function defining every link :

This rule discards all classes involving a property implemented with a missing link. The class defined by :

```
segment(s) :- name(s,n) & string(n)
              & org(s,o) & point(o)
              & ext(s,e) & point(e)
              & lg(s,l) & real(l)
```

is discarded because the term : `equal(l,dist(o,e))` is missing.

Using these seven rules to derive the closure set of the relevant classes leads to the following five classes (instead of the potential 2^{20} classes corresponding to the ten basic segment properties and their negations) :

C1 - the class of all the segments the existence of which is known :

```
segment(s) :- name(s,n) & string(n)
```

C2 - the class of all the segments the origin point of which is known :

```
segment(s) :- name(s,n) & string(n)
              & org(s,o) & point(o)
```

C3 - the class of all the segments the end point of which is known :

```
segment(s) :- name(s,n) & string(n)
              & ext(s,e) & point(e)
```

C4 - the class of all the segments the origin and end points of which are known and identical :

```
segment(s) :- name(s,n) & string(n)
              & org(s,o) & point(o)
              & ext(s,e) & point(e)
              & lg(s,l) & real(l)
              & equal(l,dist(o,e))
              & not(conf(o,e))
```

Classes C1 to C3 include all the segments that are consistent and incomplete. Class C4 includes all the complete and inconsistent segments. Class C5 includes all the complete and consistent segments.

4. AUTOMATIC CONSISTENCY CONTROL

As shown in the preceding Sections, it is possible to characterize dynamically the class to which an object belongs from a set of classes defined automatically. This section details the rules for the control of the designers operations. They are defined using the notions of completeness and consistency of the objects involved. Together with the rules defined in Section 3, they implement an automatic consistency control of the objects being modified. They are implemented as part of the system's standard capabilities.

The partial order relation between classes defined in section 2.3 is refined to take into account the **consistency degree** and the **completeness degree** of the objects.

> *Definition* : An equivalence class C1 is **smaller** than a class C2, noted C1 < C2 iff the objects belonging to C1 satisfy less constraints or have less instantiated properties than objects in C2. C2 is then **larger** than C1.

The total number of satisfiable formulae in C1 will therefore be smaller than in C2. Classes C1 and C2 defined in Section 3 above are such that : C1 < C2, because the definition of C2 includes the term : `org(s,o) & point(o)` which is not in the definition of C1.

```
C1 : segment(s) :- name(s,n) & string(n)
C2 : segment(s) :- name(s,n) & string(n)
                   & org(s,o) & point(o)
```

In the following we refine the notions of completeness and consistency of objects to implement the certification of operations. We define for this purpose the **degree of completeness** and the **degree of consistency** of objects in **CADB**.

4.1 Degree of completeness of an object

Let N be the total number of properties of an object, i.e the number of atomic first-order formulae in its specification rules. We say that an object is P-complete or has a degree of completeness P iff it bears P instantiated properties (1<=P<=N). The number of undefined properties is therefore N - P. Objects belonging to class C2 in Section 3 are 2-complete : their name and origin point are known.

> *Definition* : An object is **complete** iff it is N-complete.

4.2 Degree of consistency of an object

Let M be the total number of constraints defined for an object. We say that an object is L-consistent or has a degree of consistency L iff the total number of undecidable or satisfied constraints is L. The object therefore bears M - L unsatisfied constraints.

Recall that unsatisfied constraints (i.e decidably false) define object classes. The notion of consistency degree defined here is therefore **weaker** than the notion of satisfiability of formulae in first-order logic. It allows taking the incompleteness of the design objects into account when checking their consistency without impeding their incremental design.

Objects in class C4 above (Section 3) are 0-consistent because the only existing constraint : `not(conf(o,e))` is violated. Objects belonging to classes C1 to C3 and class C5 are 1-consistent because the constraint : `not(conf(o,e))` is undecidable or satisfied.

> *Definition* : An object in **CADB is consistent** iff no decidable constraint is violated : it is M-consistent.

4.3 Defining certification

The following rules (R8 to R10) are stored in the system's knowledge base. They define the legal operations with respect to the degrees of completeness and consistency of the objects involved and the partial order between classes.

R8 - update operations on the properties of objects are certified if the consistency degree of their prototype does not decrease :

Updates may change the objects equivalence class. This rule permits a consistent modification of objects while guaranteeing that their consistency degree does not decrease.
Stated otherwise, after an update, objects must remain in the same class or belong to a larger class with respect to the order relation "<" between classes.

If Ci(Pi,Li) is the class Ci of objects that are Pi-complete and Li-consistent, an update of an object belonging to C1(P1,L1) is

certified if its prototype belongs after the update to C2(P2,L2) where :

P1 = P2 and L1 <= L2.

Changing a segment from the class C5 to the class C4 is prohibited. The origin of a segment in C5 cannot be modified if the constraint : `not(conf(o,e))` no longer holds.

R9 - property instantiations are always certified :

CAD systems provide iterative trial and error cycles. Designers are therefore allowed in **CADB** to augment object properties without immediate consideration for their consistency.

Intantiating properties of an object in C1(P1,L1) is certified if its prototype belongs afterwards to a class C2(P2,L2) such that :

C1(P1,L1) < C2(P2,L2) and P1 < P2.

For example segments may change from class C2 to class C4. The end point of a segment in C2 may be instantiated even if the constraint : `not(conf(o,e))` does not hold. The same case applies for classes C2 and C5.

R10 - object properties may be deinstantiated if the consistency degree of the object increases :

> This rules complements rule R10. If property instantiations result in constraint violations, the designers are allowed to **undo** the instantiations.
>
> The update is certified if the new class C2(P2,L2) is such that :
> C1(P1,L1) < C2(P2,L2) where : P1 > P2 and L1 < L
> Changing segments from class C4 to class C2 or C3 is allowed (deinstantiation of origin and end points) if their consistency degree increases.

These rules are examples of expert knowledge that should be invoked to adjust the certification to the consistency controls in specific application domains.

Their usefulness comes from their ability to detect **a priori** the potential inconsistencies resulting from the update operations and to automatically compute their side-effects on the object prototypes. This is a sharp distinction with currently available data and knowledge base systems (as well as current design methodologies) since these can evaluate the object completeness and consistency degrees only after the update operations have been performed on the data items. Such features are implemented here as part of the system's expertise and provided to the designers as standard capabilities.

5. CONCLUSION

A new approach for controlling automatically the consistency of dynamic database objects is presented. It uses both the object representation models and application independent knowledge. In contrast with usual methods, it does not require the exhaustive control of the operations on the entire database content, nor the characterization of the minimal subset of objects affected by an update operation.
It relies on the dynamic characterization of the objects equivalence classes, defined by the maximal conjunctions of constraints they actually comply with. Specific representatives for these classes are used, called the object **prototypes**, on which the operations are first applied. This allows the detection **a priori** of the potential inconsistencies resulting from the designers operations on the desired objects. It also allows the automatic computation of the operations side-effects.
Heuristic rules are defined to optimize the dynamic characterization of the object relevant classes. Operating rules are also detailed that define the certification of the update operations. Altogether, they are implemented as part of the system's standard features thus enhancing its ability to manage dynamically evolving design objects. They also support **expert database** capabilities that assist the designers when creating or modifying the object structures [13].
An update operation is **certified** if its application on the prototypes of the objects involved does not **decrease** their consistency degree. Further, it also takes into account the incompleteness of the design objects in a unified framework. This is a **pessimistic** approach to consistency control in data and knowledge bases.
It is implemented in **CADB**, a prototype Expert Database System for CAD. **CADB** includes two components. A relational database system that maintains the application data and that is used as a back-end data repository and a knowledge base system that maintains the application specific knowledge. It is implemented in Prolog on APOLLO^{TM} workstations. A prototype in OPS5 is also currently designed.
Further research include extensions towards multiple representations of objects and managing non certified operations.
The first objective is to take into account the side-effects of the update operations on the diverse representations of the objects through the prototypes of their respective classes.
The second is to cope with non certified operations that the designer wishes to process although consistency controls have failed. This involves handling exceptions in data and knowledge bases [3].
Further investigations are still needed and are currently being worked out in this area.

REFERENCES

[1] ADIBA M., NGUYEN G.T.
Information processing for CAD/VLSI on a generalized data management system.
Proc. 10th International Conference on Very Large Data Bases.
Singapore. August 1984.

[2] BATORY D.S., KIM W.
Modelling Concepts for VLSI CAD Objects.
ACM Transactions on Database Systems.
Vol. 10, Num. 3. September 1985.

[3] BORGIDA A., MITCHELL T.M, WILLIAMSON K.
Learning improved intègrity constraints and schemas from exceptions in databases and knowledge bases.
On Knowledge base management systems.
Brodie, Mylopoulos Eds. Springer Verlag. 1986.

[4] FAUVET M.C, RIEU D.
CADB : un systeme de gestion de bases de donnees et de connaissances pour la CAO.
Proc. MICAD '87 Conf. Paris (France). February 1987.

[5] JULLIEN C., LEBLOND J., LECOURVOISIER J.
A database interface for an integrated CAD system.
Proc. ACM/IEEE 23rd Design Automation Conference.
Las Vegas (Ne). July 1986.

[6] KATZ R.H., ANWARRUDIN M., CHANG E.
Organizing a design database across time.
On knowledge base management systems.
Brodie, Mylopoulos Eds. Springer-Verlag. 1986.

[7] KERSCHBERG L.
Proceedings of the 1st International Conference on Expert Database Systems.
L. Kerschberg Ed.
Charleston (South Carolina). April 1986.

[8] KIM W. et al.
A transaction mechanism for engineering design databases.
Proc. 10th International Conference on Very Large Data Bases.
Singapore. August 1984.

[9] LORIE R.A, PLOUFFE W.
Complex objects and their use in design transactions.
Proc. ACM/IEEE Database Week.
San Jose (Ca.). May 1983.

[10] NGUYEN G.T et al.
A high level interface for an local network database system.
Proc. INFOCOM Conference.
Las Vegas (Ne.). March 1982.

[11] NGUYEN G.T.
Semantic data engineering for generalized databases.
Proc. 2nd International Conference on Data Engineering.
Los Angeles (Ca.). February 1986.

[12] NGUYEN G.T
Quelques fonctionnalites des bases de donnees avancees.
PhD Dissertation. Universite de Grenoble. June 1986.

[13] NGUYEN G.T, RIEU D.
Supporting dynamic database objects.
Paper submitted for publication.

[14] QIAN X., WIEDERHOLD G.
Knowledge-based integrity constraint validation.
Proc. 12th International Conference on Very Large Data Bases.
Kyoto (Japan). August 1986.

[15] RIEU D.
Modele et fonctionnalites d'un SGBD pour les applications CAO.
PhD Dissertation. Institut National Polyechnique de Grenoble. July 1985.

[16] RIEU D., NGUYEN G.T.
Semantics of CAD objects for Generalized Databases.
Proc. ACM/IEEE 23rd Design Automation Conference.
Las Vegas (Ne). July 1986.

[17] VASSILIOU Y.
Knowledge-based and database systems : enhancements, coupling or integration ?
On Knowledge base management systems. Brodie, Mylopoulos Eds.
Springer Verlag. 1986.