12

An object-oriented approach for engineering design product modelling

W. Shen, J.-P. Barthès
CNRS URA 817 HEUDIASYC
Université de Technologie de Compiègne
BP 649, 60206 COMPIEGNE, FRANCE
Tel: (33) 44 23 44 23 Fax: (33) 44 23 44 77
E-mail: [wshen | barthes]@hds.univ-compiegne.fr

Abstract

Engineering design, as one of the most challenging application areas for Computer-Aided Engineering, requires the representation of complex data elements as well as complex relationships among them. Users in these environments have found traditional technology inadequate in terms of flexibility, modelling power and efficiency. Thus, it becomes necessary to develop environments capable of supporting the various paradigms used in large projects. Such environments require a good data and knowledge representation, efficient programing features, adequate mechanisms for storage and concurrency control and good communications with other software systems. In this paper, we discuss the main characteristics of the engineering design products and the limitations of the current design product models. And then, we present the main features of an improved object-oriented representation model PDM and its implementation MOSS. An example is given for demonstrating the application of the object-oriented representation model PDM and the object-oriented programming environment MOSS in the field of mechanical CAD.

Keywords

Engineering Design Product Modelling, Object-Oriented Model, Object-Oriented Programming, Artificial Intelligence, CAD, Mechanical Design.

1 INTRODUCTION

Engineering design, as one of the most challenging application areas for Computer-Aided Engineering, requires the creation, and the manipulation of complex entities that are related by various semantic relationships. The descriptions of the design entities and the relationships change throughout the design process. A computer-aided environment that assists with the design of engineering artifacts should not only support the representation of the end product and the relationships between its components, but also the evolution of the components and associated knowledge over the whole life of the product.

T. Tomiyama et al. (eds.), *Knowledge Intensive CAD*

Computer-aided design environments supporting engineering design tasks have recently started to utilize central databases, as opposed to data files, as a means for integrating computer-aided design tools and applications. The introduction of a central database to the design environment leads to many difficulties that must be addressed and solved. Examples of such problems are managing the design data, maintaining the database consistency, resolving conflicting updates, and handling concurrent accesses. The database technology developed in recent years provides facilities that are useful for managing the repository of data generated in engineering design applications. However, the complexity of many design projects has rendered existing database techniques inadequate. This problem has stimulated the development of new database systems capable of supporting the design process. Research in semantic modelling, entity-relationship modelling, or object-oriented data modelling has resulted from this interest. Many object models have been proposed both in artificial intelligence and for databases. In artificial intelligence, models tend to act as prototypes and repositories for default values. Many models have been used in knowledge representation languages for a long time, e.g., KRL (Bobrow and Winograd 1977), RLL (Greiner and Lenat 1980), SRL (Fox et al 1986). Up to now, we have not found an efficient model for engineering design product modelling. The paper describes an approach for modelling information (data and knowledge) about complex engineering artifacts. This approach supports the representation of the components of design products and of relationships among the components, as well as the evolution of these components, their interrelationships and the associated knowledge during the design process.

2 CHARACTERISTICS OF THE ENGINEERING DESIGN PRODUCTS

In this section, we discuss the main characteristics of the engineering design products. We argue that such characteristics should be considered in developing any semantic representation of design product models. Although the examples used in this discussion are taken from the domain of mechanical design, the characteristics discussed here can be generalized to other fields of engineering design, such as architectural, or chemical engineering.

2.1 Hierarchical data

Design products are usually composed of many design entities which are either composite or primitive. A data model that can capture this recursive definition of design entities should include explicit semantic constructs for representing design assemblies. Such semantic constructs as aggregation and generalization become very useful as the design products become more and more complex.

Consider for example a gearbox in the domain of mechanical engineering (Shen et al 1994). A gearbox is composed of several design entities: a top-case, a low-case, some shafts, gears, bearings, and fastening assemblies. Each entity is itself composed of several parts related each other. Indeed, a gearbox is itself a part of a mechanical system such as a car, or a tractor.

2.2 Semantic relations

Design entities are related to each other by various relationships which are semantically meaningful and should be represented explicitly by semantic constructs. Such relationships are as important as the entities themselves in determining the behavior of the product. The traditional relational data model has been criticized for its lack of semantic constructs for expressing object interrelationships.

The entities of the gearbox mentioned previously are related to each other by spatial links such as adjacent-to, or below, and functional links such as engaged-with, or fixed-by. The fact that a gear A is engaged-with another gear B means that gear A transfers a certain load to gear B. This information represents the semantics of the "engaged-with" relationship. If such a

relationship is represented as buried pointers between objects, the semantics of this relationship will be lost. Therefore a design product model should support the explicit representation of both the existence and the semantics of a given relationship.

2.3 Different viewpoints and consistency

Different views of a design product must be generated and maintained. A representation of design product models should be flexible enough to allow the incremental generation and manipulation of such views. One should be able to generate and manipulate a structural view model of a bridge independently of other models such as the physical model and the functional model. The consistency between the various views however, can be temporarily violated during the design process. Such violations are common practice among designers.

For example, during the design of a gearbox for a car, a designer estimates the weight of the gearbox so that the weight is added to the total load of the car. When the designer finishes the design of the gearbox or modifies the design, the estimated weight is usually not equal to the actual weight of the gearbox, resulting in data inconsistency. The designer may or may not choose to immediately update the value of the total load of the car.

2.4 Multi-type or multimedia

Various data types and knowledge formalisms are used to describe the products. Numeric, symbolic, and graphical data are examples of such data types. Declarative and procedural knowledge constructs are examples of the knowledge formalisms that designers use to represent, manipulate, and reason about design information. The diversity of the data types that describe the design products and the knowledge sources that reason with this data create the need for a flexible environment that supports various data types and multiple knowledge representation paradigms.

2.5 Versions

To represent the evolution of a design, one must not only record the current state of a design object, but also some of its past states. A scheme for organizing the version of design objects becomes essential in a data model which must capture the evolutionary nature of engineering design.

In harbor preliminary design (Monceyron and Barthès 1992) sometimes up to 40 different versions are studied in parallel, leading to 2 or 3 detailed studies. In the automotive industry, several models of the same car are designed simultaneously sharing a number of features. Many more examples requiring versions can be found in design of complex electronic gears, in the development of large software package (CASE), or in the development of large knowledge bases. Such activities show a need for systems capable of handling versions which evolve in time, and which very often require the study of several hypotheses in parallel. Engineers need software environments containing the appropriate supporting mechanisms.

2.6 Constraint propagations

The design process is mainly driven by design constraints which play an essential role in the evolution of design products. There are usually different types of design constraints that require different types of actions to be taken upon their violation. For example, violating stress constraints may motivate a redesign of the mechanical system, whereas violating length constraints may be tolerated. Therefore, it is necessary to separate the representation of the design constraints from the evaluation and enforcement mechanisms of such constraints. Design constraints are used to enforce the legal compositions of assembly objects, to check the correctness of values assigned to the attributes of objects (and to enforce the correctness if possible), and to represent integrity and referential rules. Constraints that are defined as

relationships among attributes of the same class or of different classes can be either simple algebraic expressions or complex evaluation procedures. A flexible constraint representation mechanism should allow for instance-specific constraints as well as class-defined constraints, violable and non-violable constraints, and should also handle exceptions (constraint relaxation).

2.7 Meta-information

Meta-information is the information about the various pieces of data, or links in the systems. It is very useful in mechanical engineering design, e.g., for expressing the technical descriptions of the parts or of the data, the time when the information was recorded, the life time of the information, and so on. However, the problem of expressing meta-information is more complex that can be thought. A major problem results from the use of annotating objects. Indeed, in environments with potentially millions of objects, one cannot afford to associate many annotating objects with a given object, because the information will end up being scattered on disk, and will become very expensive to retrieve at a later stage. Mechanisms to prevent such scattering are also complex and expensive. Thus, a compromise has to be found between what must be recorded within the object itself, and what can be recorded as annotating objects.

2.8 Integration of graphics

A graphical interface is imperative for an engineering design environment. Recently, graphics tends to be separated from the environment and run as a separate process on a specific workstation. Objects are drawn for displaying the artifact. Input can also use parameters from the graphical picture.

2.9 Distributed processing

Real world concurrent engineering design projects require the cooperation of multidisciplinary design teams using sophisticated and powerful engineering tools. The individuals or the individual groups of the multidisciplinary design teams work in parallel and independently with different engineering tools located on different sites. In order to ensure the coordination of the design activities of the different groups or the cooperation among the different engineering tools, an efficient distributed design environment has to be developed (Cutkosky et al 1993, McGuire et al 1994).

2.10 Integration with manufacturing and project planning

The integration of the engineering design environments with manufacturing systems or project planning systems does not require new technology, but many difficult problems remain to be solved.

The main objective of the research described in this paper is to develop an improved representation to support the essential characteristics required for designing products.

3 AN OVERVIEW OF DESIGN PRODUCTS MODELLING

It has been recognized that simple conceptual data models, such as the relational model, have great advantages for applications where the information can be described in terms of a small number of types related in a well-defined way such as in business applications (inventory or flight reservation databases). However, such models lack the appropriate constructs and mechanisms to handle the complex semantics of more advanced applications such as engineering design. For example, the relational model does not provide any mechanism to

distinguish among the different kinds of relationships which may exist between entities. Such a mechanism is vital for design applications where design components are related to each other by various relationships. It will be useful, for example, to be able to describe an existence-dependent relationship where the existence of design parts is dependent on the existence of the assembly which is composed of such parts. Deleting the assembly from the database would cause the deletion of all its dependent parts, provided that such parts are not components of other existing assemblies. A design product data model which provides a rich set of semantic constructs to represent engineering design products is highly desirable. This data model should provide: i) a natural way of describing complex objects and relationships; ii) a way to describe partial descriptions of design objects and incrementally evolve these descriptions; iii) a way to describe meta-information about the design objects; and iv) an easily extensible schema that can be uniformly accessed with the data. Current research in semantic data model attempts to provide semantic constructs that support such requirements.

3.1 Categories in design product modelling

Research in design product modelling can be broadly classified into four main categories:

- *Developing general-purpose semantic data model that will overcome the limitations of traditional data models such as the relational model* (Abrial 1974, Chen 1976). Although they are not directly oriented towards engineering design information, in many cases they were motivated by the demands of such applications.
- *Developing semantic data model for engineering design applications* (Katz 1985, Yu et al 1986, Veth 1987, Tomiyama et al 1991). A key issue in such efforts was the selection of appropriate constructs and data flow mechanisms to capture the design product information. A consensus among researchers in this area is the need to develop a minimal set of information structures that could be used to build specialized information concepts in a particular design domain (Eastman et al 1991a, 1991b).
- *Identifying the specialized features of particular design domains and developing special-purpose information constructs and flow mechanisms that can be used to represent design products in applications within that domain.* Such efforts typically relied on existing information modelling techniques to support the modeled applications (Ancona et al 1990, Ma et al 1993, Zamanian et al 1992).
- *Defining the schema of a design product within a specific design domain for the purpose of structuring a large database to be used by applications within that domain.* The widest development in this area today is the STEP/PDES activities to standardize the schemas for design products in many different areas of engineering design (Bjork and Wix 1991).

Because the research in the area of design product modelling is so extensive, we will not attempt to describe any individual effort in this paper. Instead, in the following paragraphs, we observe common characteristics of the various proposed design product models and discuss the limitations inherent in such characteristics.

3.2 Limitations of the current design product models

Since a data model is defined to be a collection of structural abstractions and a set of operators to create and manipulate such abstractions, it is desirable that operators in a data model be able to consistently manipulate semantics included in the structural abstractions. For example, the create-entity operator should reject a request to create an entity which is defined to be existence-dependent on another entity unless this entity already exists in the database, or it is should automatically create such an entity if it does not already exist, or else, it should produce a warning. Similarly, a delete-entity operator should check if this entity is involved in any

relationships with other entities before taking the actions which are implied by deleting the entity.

Almost all design product data models use a common approach found in object-oriented data modelling. This common approach presumes the existence of design templates (classes or prototypical instances) that describe a design product before an instance of this design product can be created. The schema in such data model is usually defined prior to populating the database. The various classes to which design objects belong have to be predetermined. However, in most design situations, the attributes of a design entity change throughout the design process. In addition, the designer may wish to specify and view a design entity at multiple levels of abstraction.

Further, the current design product models suffer from other limitations:

3.2.1 No automatic class organization

It is not possible to take two class definitions and decide if, based on those definitions, one class is more general than the other. That is, it is not possible to determine that the set of instances of one class would in all events include all the instances of the other.

3.2.2 No automatic instance classification

It is not possible to decide if an object can be classified as an instance of a class if it has not been a priori declared as such, meaning that the user always has to declare an object to be an instance of a predefined class, rather than just focussing on refining the attributes and properties of the object and letting the model take care of the classification.

3.2.3 Limited ability to represent partial descriptions of objects

Models that are based on core object-oriented concepts exhibit limited evolution. The attributes of an object are fixed once its class is determined.

3.2.4 Limited representation and checking capabilities for constraints

Constraints are usually specified only in terms of the values of attributes of classes and their types, plus additional existence and dependency constraints in some advanced models.

3.2.5 Ability to only represent "necessary properties" of classes of objects, but not "sufficient properties"

This is a direct consequence of the fact that an object has to be explicitly declared to be an instance of a certain class, and therefore has to have all the properties which were defined a priori for this class. There is no explicit mechanism to define a class intentionally. That is, there is no mechanism to declare that an object is an instance of a specific class if it exhibits certain properties.

3.2.6 Limited multi-typing capabilities

In general, the only way to make an object be an instance of more than one class is by creating an artificial class and make it a subclass of those classes (a case of controversial multiple inheritance). It is always desirable in programming languages to provide a unique, or at least principal type for every value or expression, and therefore, programming languages do not allow a value to have multiple types. However, in a model of a design product, we would like to associate several incomparable types with an object such as a material type and a geometric type. This issue has been raised also in information modelling in general, where it has been

recognized that there is a need to represent an entity which can be both a student and a teacher (teaching assistant) without having to define a new class of TEACHING-ASSISTANT.

3.2.7 Limited evolution of state

In general, a new object is created and bound to the old identifier of the old object (some object-oriented languages such as CLOS provide a change-class operator). During the design process, it is desirable that asserting new information cause an object to be classified under new classes and retracting old information lead to the conclusion that an object is no longer an instance of a particular class.

3.2.8 No deductive or reasoning capabilities

The database (or knowledge base) plays the role of a passive repository of data and does not play an active role in deducing relationships between classes or between classes and objects.

In almost all existing object-oriented data models, a definition of a class usually contains all the necessary conditions that an instance would satisfy once it is declared to be an instance of this class (i.e., all the attributes and operations associated with the class). All "conditions" and "constraints" are defined at class level (schema definition stage) and automatically apply to the instances which populate the computer memory or the database. As stated previously, this type of approach for modelling the world of discourse proved to be successful in many domains where the schema is static and the structure of the world is not changing. However, in the domains such as engineering design, the structure of the world is constantly changing and, therefore, the design data model should be dynamic (i.e., allowing for schema evolution). For example, if the template of a design part used frequently in a design product model are changed then the properties describing the object in the product model that are declared to be instances of that design part need to be changed subsequently. In addition, models for representing knowledge cannot be structured a priori.

4 OUR APPROACH FOR MODELLING DESIGN PRODUCTS

We have tried to develop a model for meeting all the above mentioned requirements for engineering design and overcoming the limitations found in most object models. We did so by considerably improving a basic semantic model, PDM, which was developed many years ago and had enough potential to support our approach.

4.1 An overview of PDM

PDM was first proposed around 1977 at University of Technology of Compiègne (Barthès et al 1979), and was influenced by Minsky's frame model (Minsky 1975), and Abrial's semantic model (Abrial 1974). The main feature of PDM is the existence of independent properties, themselves objects, shared by any number of other objects. This approach was motivated by the early use of the representation in the domain of robotics, where objects may have well defined properties—acquired through various sensors—but do not belong to any specific class until they are fully recognized. Thus, PDM can represent objects as soon as properties exist.

PDM was also used for developing an object-oriented database VORAS which was one of the earliest object-oriented databases, and then was used in many projects, prototypes, or commercial products such as G-BASE™* and MATISSE™**.

Here we list very briefly some important specific features useful for engineering design.

---

* G-BASE was a trademark of GRAPHAEL.

** MATISSE is a trademark of ADB.

PDM distinguishes between the properties which have values directly attached to them, called *terminal properties*, and those linked with other objects, called *structural properties*. The terms stand for historical reasons and were proposed in 1977; today, following the OMG recommendations, we should call them *attributes* and *relationships*.

PDM uses a *four-layer specification hierarchy*, i.e., instance layer, model layer, meta-model layer and meta-meta-model layer. As usual in this kind of approach, the object at the top-level (meta-meta-model) is its own specification, i.e., there is no meta-meta-meta-model layer. Thus, a *structural reflectivity* is achieved at this level. The two higher levels define the internal structure of the chosen representation, and constitute the kernel of the system.

Reflectivity is a powerful concept, since it allows to modify dynamically the structure of the model itself, which is very useful in the engineering design environments as mentioned in the previous sections; however it is difficult to handle. *Reflectivity* is also an extremely useful feature for automatically providing explanations, or for implementing learning mechanisms.

At all model levels, in particular at the class level, PDM factorizes information in class hierarchies. In practice, a class may inherit information (structure, defaults and associated knowledge) from several other classes through an IS-A structural property, which can overcome the limitation 3.2.1.

An interesting feature of PDM is the possibility of defining IS-A links between instances, thus introducing a *prototyping* mechanism (Figure 1). This can be used in particular at property level for particularizing a more general property.

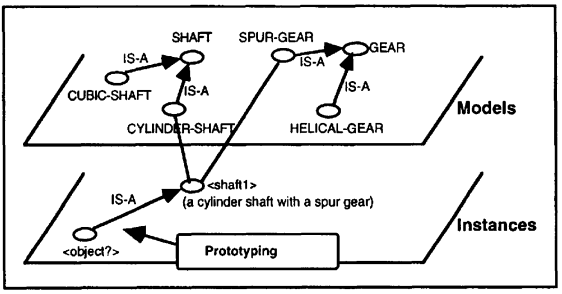

Figure 1 Inheritance hierarchy within a specification layer.

Another important feature is the possibility to create *classless objects*. In Figure 1, we have an object "object?" which is not an instance of any class, but which seems to have the same function and responsibility as "shaft1". So we assign an IS-A link between "object?" and "shaft1", and "shaft1" then acts as a prototype for "object?". The concept of classless objects is useful for recognition processes, e.g., scene analysis in robotics. It can be used together with automatic instance classification.

In PDM, it is possible to overcome the limitations 3.2.3 and 3.2.6 by using the mechanism for classless objects, i.e., we can define an object "a cylinder shaft with a spur gear" from the properties of the class CYLINDER-SHAFT and the class SPUR-GEAR without defining a new class.

Contrary to what is found in most models, in PDM values associated with terminal properties are not restricted to any particular type, and can be anything from a string to an image or an executable function (which is useful for implementing *demons*). However they can be restricted by using the demons attached to the specific terminal property thus implementing data types. Such *demons* are invoked at creation or modification time (as mentioned in the section 3.2) or for displaying values.

In PDM, each entity (all objects, classes and instances) has an internal unique identifier, its internal key. Any objects can grow or shrink at any time. Classes are modeled as objects

providing an integrated data dictionary. PDM is a consistent object system, i.e., every object has a unique identifier, and there are no dangling references.

Terminal properties may be inverted, creating external references to objects called *entry-points*. Such inversions occur at creation time, and are performed by an appropriate demon. Entry-points are automatically maintained and updated when values are modified. Entry-points are not required to be unique, which simplifies indexing. Comparison operators may be user-defined and associated with terminal properties. Structural properties are automatically inverted, which allows one to navigate easily within the database, and avoids extra indexing structures. Such a feature is imperative in the engineering database management systems.

Introducing *versions* allowed us to represent temporal changes or parallel hypotheses, and had a strong impact on the concurrency mechanisms. The versioning mechanism is very useful in the domain of CAD for stocking the modifications during the design process (Borhani et al 1992). Versioning can also be used for exploring parallel hypotheses, or for doing concurrent design, letting design teams diverge, then reconciling the separate designs (Barthès 1993).

Traditionally, *default values* when present are associated with attributes or with slots in frames (using the default facet). When an attribute is a concept that can exist independently of any class, then we have to use a different approach. Thus we cannot attach default values directly to attributes. In PDM, two mechanisms were successively tried to cope with defaults: (i) extending the concept of property, which led to *property-tree* (see next subsection); (ii) introducing the new concept of *ideal instance*. We modified PDM to associate with each class an *ideal instance* which contains default assignments. Thus, if a value cannot be found for an object, either locally, or after following the prototype links, then, if the object belongs to a class, the *ideal instance* is accessed and the default value if present is retrieved. The mechanism was introduced recently and works well in simple cases, but needs to be tested for more complex ones.

4.2 Implementation of the model PDM

Choosing a data structure for implementing the PDM representation involved several constraints: (i) the dynamic aspect of the objects; (ii) the capacity for a large number of objects; (iii) the problems of object sharing and distribution. The environment that resulted from the implementation of the PDM model is called MOSS (Barthès 1994).

Object data structure

The LISP environment provided the dynamic features with the use of lists rather than defstruct. The simple association lists were used in MOSS for allowing to store an object as a sequence of <property, list of values> pairs. Specifying the property explicitly in a LISP environment is virtually free. Traditional frame languages on the other hand express an object as a 3-level structure frame/slot/facet, where facets are used to contain sometimes meta information. Names for facets are often predefined, and characterize the resulting frame representation language. With the PDM approach, properties—corresponding to slot names—are first class objects. Thus, the property name refers to a separate object, and facets are in turn expressed as properties of that object. Hence, facets are factorized for all objects sharing the property. This is very clean and efficient in terms of storage, which is a good point for handling a large number of objects.

In MOSS all structural properties (relationships) are inverted, thus providing referential integrity. The consequences of such a choice are: (i) an easy way to navigate from an object to its neighbors; (ii) some savings in maintenance of the access paths. A disadvantage is that if one wants to use locks on objects, then neighbors have to be locked. All values associated with properties are multi-valued. Thus a person can have several names, first names, etc. Cardinality constraints can be imposed on the number of values associated with a property.

Object identifiers

The question of producing internal object identifiers is important for object systems. In MOSS we chose an approach with synthesized unique names reflecting the both class of an object (using the class radix and class sequence number), or else nothing particular for a classless object. This approach is interesting when one has to traverse a class, since it is possible to synthesize all object identifiers directly without the need for keeping a long class list. In practice, this amounts to a hash-link. The internal object identifiers are used as keys for organizing the objects on the disk.

Entry-points

In many approaches the user has no access to the internal object identifiers, and must keep references to objects in (possibly persistent) variables. To access or to locate an object, the user has to go through the variables or sometimes formulate a query. Since we want to deal with very large numbers of objects, we proposed the concept of *entry points. Entry points* are somewhat similar to keys in databases, i.e., they correspond to some value characterizing the object, with the difference that they need not be unique. In fact the *entry point* mechanism is used to invert some of the terminal properties (attributes). MOSS implements *entry points* as first class objects, whose internal names are the value of the index. The *entry point* mechanism allows to locate objects easily. An object may have any number of entry points, possibly for the same property. *Entry points* may be produced by means of special functions.

In addition, MOSS has some other important features: messages are transmitted by using functional calls (send); methods can be attached to instances as well as to classes or meta-classes; there are several types of methods, i.e. own-method that apply directly to the objects to which they are attached and instance-method which apply to the instances of the class to which they are attached; there are universal methods that are not attached to any object; inheritance is multiple and flexible for universal methods; inheritance can be redefined as user's will; methods are compiled incrementally wherever they are called to improve future calls; models can be grouped in applications, systems or sub-systems, allowing a modular approach.

In the recent years, with the progress of the research, several important features have been added to MOSS.

Meta-information and property tree

The desire to be able to add *meta-information* to structural properties (relationships) led to the concept of *property tree.* As mentioned earlier, if we stored *meta-information* at the property level, then it would be shared by all objects having this property. Thus, to associate *meta-information* with specific links, we experimented with a special way for organizing properties, i.e., using the prototyping link IS-A whenever we needed to particularize some properties.

In this way, it is possible to organize a property as a tree structure, the root of which corresponds to a generic property, with the nodes of the tree as variations on that property. Thus, one can have a generic WEIGHT property, sub-properties for GEARBOX, for GEAR, for SHAFT, each referencing the same generic property, and each having for example a different default; similarly, specific WEIGHT properties found in some instances are expressed as subobjects in the tree. This mechanism is very important in the complex engineering design projects.

Query mechanism

Queries are used to locate objects. We work as if the objects were in memory, and when there is an object fault the system tries to obtain it from the disk. Query mechanisms are imperative for engineering CAD systems and engineering database management systems. In MOSS, queries can be simple direct queries or more complex ones. Direct queries use *entry points* to retrieve all objects corresponding to a piece of data. The process is more complex than one imagines at first, since reformatting demons can interfere with indexing; the result is a set of

object-ids. Complex queries can be asked as in traditional database systems. The result is also a set of object-ids.

Persistency

For engineering design projects, in particular for large design projects, we have to handle a large number of objects and associated knowledge, and therefore we need to keep objects in secondary storage and to load them into working store only when necessary (Barthès et al 1989). However LISP environments are well known for their poor access methods (streams) obliging us to resort to back end databases for storing data. Such a solution is not well adapted when the number of objects is very large. In addition conventional relational databases show a poor behavior when one wants to add dynamically new attributes, or to create new individual links among objects.

We adopted the strategy of storing objects in a flat format contiguously on disk. The LISP environment is used as a cache for objects during transactions. A LISP/disk interface MGF (Barthès 1985) was designed to allow to read and to write a single object from or onto disk. Special disk and in-core formats were investigated so as to minimize reformatting time. A new object server is being developed for the environment of SUN workstations.

Versioning

Each object contains a set of attributes with associated values. A new version is created when a new set of values is specified for an attribute. MOSS stores the changes within the object itself, associated with a transaction stamp. This is possible due to the dynamic format selected for implementing the objects. In parallel, the relationship between stamps is stored in a version graph which records the possible states of the database, thus providing a viewpoint mechanism similar to that of ART™ (Clayton 1985) and of O2™ (Lécluse et al 1987).

The global version graph is not a tree, since recombination of versions can lead to merged branches as shown on Figure 2. In the case of merged versions, there is no ambiguity unlike in the case for multiple inheritance in semantic models.

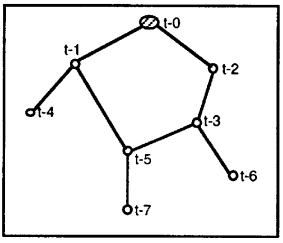

Figure 2 Version graph showing a node t-5 corresponding to merged versions.

In the MOSS system all objects are versioned, including classes, instances, classless objects, properties, methods, meta-classes and meta-meta-class. However, although consistency is enforced between classes and instances, there is no provision for a mechanism associating new classes with old instances, or new instances with old versions of classes.

In the MOSS system, the end user has no control on the version mechanism. A new version is automatically created at each new general update. A new revision can be created by collapsing all information onto a single node like t-5 in the version graph (see Figure 2). The other nodes can then be archived. The process is somewhat similar to the "believe" command in ART™.

Concurrency

Concurrency is used to allow several users to work simultaneously on the same information. A good discussion can be found in (Kim et al 1987). For multi-user CAD applications, the concurrency problem differs from that for business applications. Indeed, in business applications like banking, there are many changes - transactions - of short duration (less than one second). In CAD applications the number of users is not large, but they tend to keep large amount of data and knowledge for a long time - from a few minutes to several weeks. Hence the concurrency mechanisms are more important than that for other applications.

Traditional approach for solving concurrency problems has been to use locking. In MOSS, we took a different approach. The above mentioned versioning mechanism is associated with the concurrency control. MOSS uses an optimistic concurrency approach, i.e., when an updates is performed, all operations are checked for consistency with the previous operations since the user read context. If none of the integrity constraints is violated, then the version is added linearly to the version-graph. If some inconsistency is found, then the user is notified and may elect either to modify the data to comply with the recorded modifications, or to create an alternative database version.

Distribution

As mentioned in the section 2.9, real world engineering design projects require the cooperation of multidisciplinary design teams using several sophisticated and powerful engineering tools including several knowledge-based systems. On the other hand, each knowledge-based system may be large, complex and domain-dependent. The traditional technology such as knowledge base or centralized model (blackboard system) cannot meet the concurrent engineering requirements.

We have experimented with adding communication mechanisms into MOSS for defining cognitive agents and high-level messages, and encapsulating engineering tools by a research project OSACA (Open System for Asynchronous Cognitive Agents) (Scalabrin and Barthès 1993), which is experimented by another research project DIDE (Distributed Intelligent Design Environment) (Shen and Barthès 1994) for engineering design.

Importance of properties

In most of the representation models, all properties of the same entity (the class) have equal importance. But in the real world, this can be quite different. We have added an optional feature to MOSS for describing the importance of properties at the class level. This optional feature is useful for CAD systems (Shen et al 1994) and for database management systems.

The problem of multi-type or multimedia mentioned in the section 2.4 has been resolved by the commercial database MATISSE™ (Multimedia Advanced Technology for Information Systems Semantic Engineering) which was developed from the PDM model. It has also been considered in developing new object server of MOSS. The major objective of such development is to add the version mechanism into OODB, which does not exist in MATISSE™.

MOSS was first developed in an augmented VAX/VMS LISP (UTC-LISP) environment around 1987. At present, MOSS has four versions: Version 3.1 on UNIX (SUN Lucid Common Lisp 4.0), Version 3.2 on MacIntosh (MacIntosh Common Lisp 2.0) and Version 2.2 on VAX/VMS (VAX Common Lisp and UTC-LISP). For version 2.2 on UTC-LISP, there is an object server MGF (Barthès 1985). And for version 3.2 on MacIntosh, we use a simulated database for keeping all objects in a sequential file because the MacIntosh environment does not have a server for sharing objects among several users. For Version 3.1, we are currently using MATISSE™ for storing objects. We are also developing another new object server for MOSS.

5 A SMALL EXAMPLE IN MECHANICAL CAD

In the past ten years, we used the object paradigm to tackle various types of design problems. Results were reported in (Monceyron and Barthès 1992) for harbor design, in (Thoraval 1991) for car design, in (Gaillard 1994) for the design of mechanical parts with composite materials, in (Shen et al 1994, El Dahshan and Barthès 1988) for constraint propagation. We also experienced with environments using complex data and knowledge models (Barthès et al 1989) including sharing and persistency, and distributed approached involving cognitive agents (Ribeiro and Barthès 1992). Recently, we are developing a multi-agent platform OSACA (Scalabrin and Barthès 1993) and a multi-agent engineering design environment DIDE (Shen and Barthès 1994).

Here we will give a small example of the application of the object-oriented programming environment MOSS in the field of mechanical CAD.

5.1 Representation of the mechanical objects

Because of the important features of MOSS such as semantic representation, versioning, persistency, concurrency, MOSS is quite suitable for the representation of the mechanical objects (Shen et al 1994), the representation of the design knowledge (Barthès 1994, Finger and Rinderle 1989), and even the representation of the messages and the agents in the distributed intelligent design environment (Shen and Barthès 1994). We now present how we use MOSS in our design environment for representing the mechanical parts and for propagating the constraints.

We consider an example of a simple gearbox with two engaged gears. It is composed of many types of mechanical elements such as gears, shafts, bearings, bolts, nuts, washers, rings, sleeves and gaskets. Here we consider only some important parts such as input shaft, output shaft, gears, bearings, bolts, nuts, washers, top-case and low-case (Figure 3). This gearbox can be evidently represented as Figure 4 using semantic relations.

The definition interface of MOSS allows us to create classes simply. For example, the class GEAR can be defined as following:

```
(m-defclass GEAR $GEAR gear_box
   (:tp GEAR-MODULE)
   (:tp GEAR-NUMBER)
   (:tp GEAR-FACTOR)
   (:tp WIDTH)
   (:tp INTERNAL-DIAMETER)
   (:sp PART-OF)
   (:sp ENGAGE GEAR)
   (:sp MATCH OUTPUT-SHAFT))
```

A gear is defined with five terminal properties: GAER-MODULE, GEAR-NUMBER, GEAR-FACTOR, WIDTH and INTERNAL-DIAMETER; it is related to other objects through structural properties: composition relation PART-OF, ENGAGE with another gear and MATCH with output shaft. We can define all other classes in the some fashion. We can also define the instances of these classes by means of the macro "*m-definstance*".

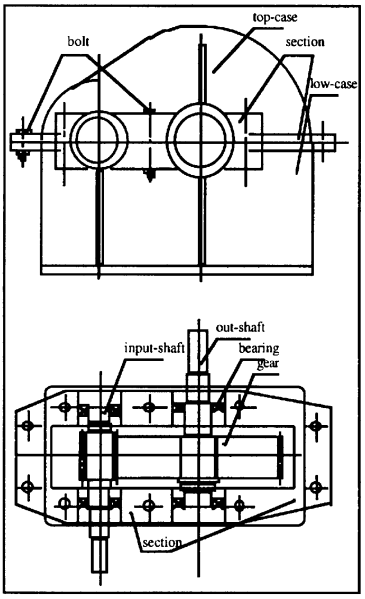

Figure 3 A simple gearbox.

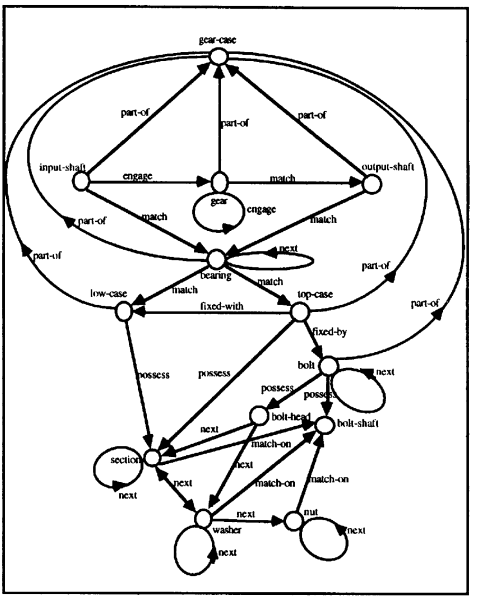

Figure 4 Semantic relations among the classes.

5.2 Knowledge representation: constraint propagation in the mechanical CAD systems

In MOSS, knowledge is represented by using the same formalism as that for data. Here we take the problem of constraint propagation as an example for showing how knowledge is associated to objects.

We have defined several methods for modifying properties of objects and for propagating constraints. We have compared the two techniques for constraint propagation, namely, along the hierarchic links and along local semantic links. We selected the later for our system due to its advantages that the components of the system can be presented clearly by using semantic links and that the constraint verification can be done while the objects are being modified (Shen et al 1994).

If we have to design or redesign this gearbox - e.g., we decide to modify the reduction rate of the gearbox and the module of the gears - then, it can be done simply by sending a message:

```
(send <gearbox> '=modify-values <new-module> <new-reduction-rate>)
```

In the system, the modification propagates first to the input shaft by a message contained in the method *=modify-values* for the class GEARBOX, and then to all other parts by messages contained in the methods *=modify-values* for the classes INPUT-SHAFT, GEAR, BEARING, and so on. For example, in the method *=modify-values* for the class GEAR:

```
(m-defmethod =modify-values GEAR gear_box (m r)
  "modify the dimensions of the gear"
  (let* (( ......))
       (send *self* '=set 'HAS-GEAR-MODULE (list m))
       (send *self* '=set 'HAS-GEAR-NUMBER (list r))
```

```
        ......
       (send (car (g==> 'HAS-MATCH)) '=modify-values  <...>)
        ......
    )
)
```

Propagation continues as new messages are sent to other objects until that all modifications are finished.

6 CONCLUSION

Most industrial applications need to handle a large amount of information which are linked by complex relationships. The traditional programming environment and database technology cannot meet such complexity. An object-oriented representation model PDM was proposed and an object-oriented programming environment MOSS was developed as a research prototype for implementing its features. In the MOSS project, we tried to build an object environment as a support for developing engineering CAD systems and other applications by integrating a good representation model, with persistency, concurrency control, and object-oriented programming. The MOSS project is a research prototype used to test various mechanisms, either original (e.g., mixing classes and instances with classless objects, inheritance by means of the three kinds of methods, ideal instances, flexible inheritance, or the specific versioning scheme).

A small example has been presented to demonstrate the application of MOSS in the domain of mechanical CAD systems, which showed that MOSS is quite suitable for developing CAD applications. Furthermore, the object-oriented technology such as object-oriented programming and object-oriented database technology, provides an effective solution for complex engineering problems.

The MOSS implementation described here refers to a research prototype. The commercial implementations of PDM such as G-BASE™, MATISSE™ use simpler mechanisms. It was found however a few hundred thousand objects could be handled easily. We believe that several millions won't be a problem.

PDM is an object representation model constantly improving and MOSS is a research prototype. There are still some problems not completely resolved, such as sophisticated query mechanisms, or natural language interfaces, which are also the perspectives for our future work.

7 REFERENCES

Abrial, J.R. (1974) Data semantics, in *Data Base Management,* 1-59 (eds. Klimbie, J.W. and Koffeman, K.L.), North-Holland.

Ancona, M., Clematics, A., De Floriani, L., and Puppo, E. (1990) HIDEL: a language for hierarchical VLSI design, *The Computer Journal*, **34(3)**, 195-206.

Barthès, J. P., Vayssade, M., Miaczynska-Znamierowska, M. (1979) Property Driven Data Base, in *Proceeding of 6th IJCAI*, Tokyo, Japan.

Barthès, J.P. (1985) MGF Version 2.0M: Moniteur de Gestion de Fichiers, UTC/GI/DI/85-12.

Barthès, J.P., Anota P. and El Dahshan K. (1989) An Experience in Adding Persistance to Intelligent CAD Systems, Texel, The Netherlands.

Barthès, J.P. (1993) La problèmatique de reconciliation en ingénierie simultanée, in *Actes de 01 DESIGN '93*, Tunis.

Barthès, J.P. (1994) Developing Integrated Object Environment for Building Large Knowledge-Based Systems, *International Journal of Human-Computer Studies*, **41**, 33-58.

Bjork, B. and Wix, J. (1991) An Introduction to STEP, Technical Report, VTT Technical Research Centre of Finland and Xix MacLelland Ltd., England.

Bobrow, D.G. and Winograd, T., An overview of KRL, a Knowledge Representation Language, *Cognitive Science*, **1(1)**, 264-285.

Borhani, M., Barthès, J.P., Anota, P., and Gaillard, F., A Synthesis of the Versioning Problems in Object-Oriented Engineering Systems. In *Proceeding of Third International Conference on data and Knowledge Systems for Manufacturing and Engineering*, Lyon, France, 165-182.

Chen, P. (1976) The entity relationship model: towards a unification view of data, *ACM Transactions on database Systems*, **1(1)**, 9-36.

Clayton, B.D. (1985) ART: Reference Manual, Inference Corporation, Los Angeles, CA.

Cutkosky, M.R., Engelmore, R.S., Fikes, R.E., Genesereth, M.R., Gruber, T.R., Mark, W.S., Tenenbaum, J.M. and Weber, J.C. (1993) PACT: An Experiment in Integrating Concurrent Engineering Systems, *IEEE Computer*, 26(1), 28-37.

Eastman, C.M., Bond, A., and Chase, S. (1991a) A formal approach for product model information, *Research in Engineering Design*, **2(2)**, 65-80.

Eastman, C.M., Bond, A., and Chase, S. (1991b) Application and evaluation of an engineering data model, *Research in Engineering Design*, **2(4)**, 185-207.

El Dahshan, K. and Barthès, J. P. (1988) Implementing Constraint Propagation in Mechanical CAD Systems, in *Proceeding of Second Eurographics Workshop on Intelligent CAD Systems*, Veldhoven, Netherlands.

Finger, S. and Rinderle, J. R. (1989) Representation of Mechanical Design, in *Proceeding of The Third Workshop on Intelligent CAD*, IFIP Working Group 5.2, Osaka, Japon.

Fox, M., Wright, J.M. and Adam, D. (1986) Experiences with SRL: An analysis of a frame-based knowledge representation, in *Expert Database Systems*, (ed. Kerschberg, L.), The Benjamin/Cummings Publishing Company Inc., 161-172.

Gaillard, F. (1994) Sur la modèlisation des connaissances et l'utilisation de bases de données objet en productique, Thèse de Doctorat, Université de Technologie de Compiègne.

Greiner, R. and Lenat, D.B. (1980) A Representation Language Language, in *Proceeding of AAAI-80*, Stanford, U.S.A., 165-169.

Katz, R.H. (1985) *Information Management in Engineering Design*, Springer Verlag.

Kim, W., Banerjee, J., Chou, H., Garza, J. and Woelk, D. (1987) Composite object support in an object-oriented database system, in *OPS 87*, 118-125.

Lécluse, C., Richard, Ph. and Velez, F. (1987) O2, An Object-Oriented Data Model, Rapport ALTAIR, Rocquencourt, France.

Ma, Y., Ye, F., Li, G., Gong, P. and Jiang W. (1993) Development of Knowledge Base/Database System with Object-Oriented Programming Environment, *Journal of the University of Petroleum of China*, **17(Suppl.)**, 299-309.

McGuire, J., Huokka, D., Weber, J., Tenenbaum, J., Gruber, T., and Olsen, G. (1993) SHADE: Technology for Knowledge-Based Collaborative Engineering, *Journal of Concurrent Engineering: Applications and Research*, **1(3)**.

Minsky, M. (1975) A framework for representing knowledge, *The psychology of computer vision*, (ed. Winston, PE.), MacGraw Hill.

Monceyron, E. and Barthès, J.P. (1992) Architecture for ICAD Systems: an Example from Harbor Design, *Revue Sciences et Techniques de la Conception*, **1(1)**, 49-68.

Ribeiro Gouveia, F., and Barthès, J.P. (1993) Cooperative Agents in Engineering Environments, in *Proceeding of EuropIA'93 Workshop on Intelligent Information Systems*, Delft, Netherlands.

Scalabrin, E., and barthès, J.P. (1993) OSACA, une architecture ouverte d'agents cognitifs indépendants, in *Actes de la Jounée "Systèmes Multi-Agents"*, Montpellier, France.

Shen, W., Barthès, J.P. and EL Dahshan, K. (1994) Propagation de Contraintes dans les Systèmes de CAO en Mécanique, *Revue internationale de CFAO et d'infographie*, **9(1-2)**, 25-40.

Shen, W. and Barthès, J.P. (1994) A Distributed Architecture for Design Environement Using Ashnchronous Cognitive Agents, in *Proceeding of Second Singapore International Conference on Intelligent Systems*, Singapore.

Thoraval, P. (1991) Systèmes intelligents d'aide à la conception: ARCHIX & ARCHIPEL, Thèse de Doctorat, Université de Technologie de Compiègne.

Tomiyama, T., Xue, D., Ishida, Y. (1991) An Experience with developing a Design Knowledge Representation Language, in *Intelligent CAD Systems II: Implementation Issues,* (eds. Ackman, V., ten Hagen, P.J.W., Verkamp, P.J.), Springer Verlag, 130-150.

Veth, B. (1987) An Integrated Data Description Language for Coding Design Knowledge, Report CS-R8731, CWI, Netherlands.

Yu, X., Ohbo, N., Masuda, T. and Fujiwara, Y. (1986) Database Support for Solid Modelling, *The Visual Computer*, **12(6)**, 358-366.

Zamanian, M.K., Fenves, S.J., and Gursoz, E.L. (1992) Representing spatial abstractions of constructed facilities, *Building and Environment*, **27(2)**, 221-230.

8 BIBLIOGRAPHY

Weiming Shen obtained his BS and MS in Mechanical Engineering from Northern Jiaotong University, Beijing, P.R. China in 1983 and 1986. He worked at Northern Jiaotong University as a lecturer in the department of mechanical engineering from 1986 to 1992 and was appointed as the vice-director of CAD/CAM Laboratory by the university in 1989. Currently he is a PhD Student at University of Technology of Compiegne, France. His research interests are design-knowledge representation, development of object-oriented environments, distributed artificial intelligence and its applications in engineering design.

Jean-Paul A. Barthès obtainted his PhD at Stanford University in 1973. Currently he is a professor in the department of computer science at the University of Technology of Compiègne, and is in charge of the research in Artificial Intelligence. His main interest lies in distributed artificial intelligence (societies of cognitive agents) with applications to the domain of current engineering. He is also the president of IIIA, an institute hosting a European industrial program concerned with the problems of capitalizing industrial and corporate knowledge.