Paper published in: New Advances in Computer Aided Design & Computer Graphics, X. Zhang, Ed., International Academic Publishers, Beijing, China, 1993, pp. 368-373

A Multiple Perspective Object Oriented Model for Engineering Design[1]

G. Pierra
Laboratoire d'Informatique Scientifique et Industrielle - ENSMA
20 rue Guillaume VII - 86034 Poitiers Cédex
Tél. (33) 49-60-50-50 - Fax : (33) 49-60-50-00
e-mail : PIERRA@ENSMA.UNIV-POITIERS.FR

Abstract

When using the object oriented paradigm to model real world objects, the perspective adopted by the modeller influences greatly both the class hierarchy (classification) and the features defined at the class level (characterization). This problem is crucial in engineering design which involves very different varieties of experts and where the use of the scientific methodology, based on simulation through "models", leads to distinct kinds of characterizations for the same technical object. Besides the name conflict problem, and the combinatorial explosion of the number of classes, multiple inheritance makes it difficult, for one expert, to work with a single perspective. In particular, when creating an instance, he must deal with a vast amount of information concerning other disciplines.

We propose to introduce the is_view_of relationship. It consists in modelling real world objects through two types of class simple hierarchies. The general_model class hierarchy carries only the properties that identify a technical object and whose perspective-specific properties may be derived by means of static data structures. The set of functional_model class hierarchies carries the different categories of representations that correspond to the different perspectives.

The is_view_of relationship is an instance connection that shall be supported by the software system. When a functional_model class is_view_of a general_model class, this means that for each instance in the latter there exists one instance in the former that is a "model" of it, and which provides services corresponding to one perspective. The concepts of category and level of representations are also discussed.

Introduction

The engineering design process is the process of specifying, designing, simulating, manufacturing and maintaining technical products that fulfill some specific requirements. This process is basically an information process [SYV 92], and computer systems are extensively used to support it. The generic term Computer Aided Engineering systems (CAE) may be used to denote these systems.

[1] This research was partially supported by the French *Ministère de l'Industrie et de l'Aménagement du territoire* (grant 90.1.3134).

The design process is a complex and iterative process that involves several experts with different perspectives about the product. Hence, the CAE systems should provide efficient mechanisms and tools to permit each expert to work on its own perspective, while ensuring coordination and feedback between the various perspectives.

In several engineering design fields, the technical products to be designed are essentially assemblies of preexisting technical objects. It is the case, for instance, in electronic [FLA 92], in piping [SYV 92], but also in mechanics [GUI 89], particularly in machine design [MOR 86]. In such fields, the CAE systems shall provide libraries of computerized representations of these preexisting technical objects, in order to avoid to define the same object repeatedly. Each expert category having its own perspective on the technical object, several representations are to be available in a library to represent, e.g., its mechanical properties, its cost, its geometry, its electric behavior and so on. Some of these representations may be described by simple data (e.g., the price, the supplier...) but other ones either require (e.g., transfer function in analogic simulation model), or generally use (e.g., parametric program for geometric representation) both data and programs. When one designs a product model, technical objects are selected in the libraries and various representations of them are to be created inside the modelling structure of the CAE system, and then to be managed by it.

In recent years, the Object Oriented Paradigm (OOP) has emerged as a pervasive and useful concept in many areas of computer science. Its flexibility and expressiveness are easily demonstrated by the wide spectrum of applications based on the concept of object, including programming language [GOL 83] [STR 86] [MEY 88], database management system [ATK 89] [KIM 90], knowledge-based systems [MAR 90]. For modeling technical objects, OOP provides two very attractive features. First, its class/instance modeling approach allows for the factorization of the description of cognate technical objects, for instance family of parametrized components. Second, by gathering data and programs, its instance concept permits to model any king of representation, wether it may be done by simple data, or it requires programs. Several CAE systems based on OOP recently appeared on the market [BEE 89] [BEZ 92] [TOU 92]. In spite of its power, the OOP still presents one important problem [MAR 90] [CAR 90]: it provides no adequate tool to structure an object model in different perspectives, even if the multi-expert manipulation of this model requires such a structure.

To introduce multi-representation capabilities inside a software system requires, first, that the needs about this facility be precisely stated, and, second, that adequate concepts to support these needs be available in the earlier stage of the software design process: namely during the analysis phase. It is the goal of this paper to contribute towards an extension of the existing object oriented analysis (OOA) methodology to support multi-representation of real world object.

In the first section of this paper, we discuss the requirements for the support of a multi-expert engineering design process. The focus is on the design processes that

mainly consist of assembling preexisting objects. In section 2, we review the different approaches proposed to provide multi-representation facilities in the framework of the OOP, either at the programming language level, or during the system design stage. In section 3, we study the abstraction mechanisms that play a major role in engineering design. In section 4 we propose a multiple perspective approach for modelling engineering design objects. We use the Coad and Yourdon OOA formalism [COA 91] to model this problem domain, and we introduce some extensions to support multi-representation of the same real world product. Finally, the paper discusses about some implementation issues.

1. Multi-representation in engineering design

Different interpretations have been given to the notion of perspective. Two of these are pertinent for the engineering design process.

The first one is the *multi-expert* one. A perspective of a world is the perception a category of experts of a particular discipline has of this world [MAR 90]. Each expert category (e.g., stress analysis engineer, thermal analysis expert, value analysis expert) is interested only about a small subset of the complete set of possible product characteristics. Different *categories of representation* (e.g., geometry representation, thermal simulation representation, kinematics representation) shall be provided and supported by the CAE system. A powerful multi-expert model must allow each expert to work with a simple category of representation: he must be able to focus only on the class structure corresponding to its particular perspective, and to consider for each instance only the relevant attributes [MAR 90].

The second one is the *multi-level* perspective. When dealing, for instance, with the geometric representation of a product, a mechanical designer needs different *levels of geometric representation* of the same technical object according to the precise problem he is trying to solve [MIN 86], and the stage of its design process. A simple axis line is sufficient to represent a screw in several stages of the design of a specific machine. The solid model of this screw may be needed during collision checking. 2D standard representations have to be represented in the final drafting. Structured methodologies [DEM 78] [GAN 79] [SCH 77] provide extensive examples of multilevel representations.

The difference between categories and levels of representations is based on the notion of interchangeability. Of course, categories are not predefined and their choice depends on the system designer. For instance, the kinematics joints between the components of an assembled object may be embedded inside its geometric representation, or they may be separated as a different category of representation. But, in the latter case, to exchange a kinematics representation with a geometric representation within the same product model representation is without any meaning. On the contrary, to exchange a simplified geometric model with an extended one, or even to exchange a solid geometric model with a 2D one [BEZ 92], or to exchange a thin-mesh-based finite element model with a thick-mesh-based one are both meaningful and useful.

Besides these representation categories and level distinction, the system shall also provide for "sharing". In fact, different categories of representation refer to the same technical object. The fundamental characteristics of this object, and, in particular, its identity, are of interest for all the experts and shall be observable from each category of representation. Interchangeability of various levels of representation shall be possible inside a specific product model, then some insertion characteristics (e.g., positioning, visualization attributes, whole-parts relationships) shall be shareable between the representations of the same category belonging to different levels.

The needs for multi-representation in the engineering design process may be summarized as follows.

1) Capability to characterize each representation.

2) Capability to add or to suppress categories of representation.

3) Capability to work with a single category of representation, both at the atomic technical object level and at the assembly or product level.

4) Capability to change efficiently between two levels of representation belonging to the same category.

5) Capability to share common features:
- between categories
- between levels of the same category.

2. Current multi-representation facilities in OOP based systems

2.1. Multiple inheritance

Most object-oriented systems do not provide with particular tools for managing multi-representation of objects. They models the different perspectives in a hierarchy and aggregate the different representations through multiple inheritances. Besides the problem of name conflict, extensively analyzed in [CAR 90], multiple inheritance offers no independence of the different representation embedded in the same instance. Each expert of one discipline must cope with a vast amount of non relevant information. Some systems, such as LOOPS [BOBR 87], introduce a multi-representation facility at the instance level: the attributes of the instance are split up into disjoint subsets, but this approach does not permit to share attributes between several subsets. A much more powerful approach is proposed in the ROME language. On the one hand, a class may be used to specify a perspective on one of its subclass [CAR 90]. This solves the name conflict problem. On the other hand, the capability for one instance in belonging at the same time to one I-class (which may carry the identity of the object) and to several parallel hierarchies of R-class (which provides for multiple representations) [CAR 88] makes the ROME language an integral part of the other approaches discussed in the next section.

2.2. Parallel hierarchies of classes

To ensure a greater independence between the different representations of the same object, several recent systems [MAR 90] [WOL 91] [PER 92], or models [STY 92], propose to model each perspective on a real world object as an instance of a different class.

The TROPES system [MAR 90] [MAR 91] used to represent and classify knowledge, is a frame-based system that allows to associate to the same concept various class trees corresponding to the various expert perspectives. A semantic link, called a bridge, permits to specify that two classes describe the same set of objects. This link is operated by the software systems. The set of attributes that identify a concept is shared by all the representation. The SYSTALK system [WOL 91] [PER 92] is a Smalltalk system that allows multiple perspective representations of complex assembled objects, namely robots. Each representation, called a facet, is an instance of a different class. Another category of classes, called the links, permits to build up the various representations of an assembled object from the representation of its components. The Generic Object Model (GOM) is a conceptual metamodel [SYV 92] which proposes to model technical objects through specific class instances, called subjects, which carry the identify of technical objects, and through role objects that carry the various representations of the technical objects. It also proposes to model the relationships between subjects or between role objects by instances of other categories of classes, called relationships.

The object oriented analysis and the resulting Object Model for Engineering (OME), proposed in the next sections, belong to this current of thinking. Like TROPES, ROME or GOM it distinguishes the identity of a technical object from its various representations, each one being modelled as a class instance. Like SYSTALK, both the atomic object representations and the relationships between atomic object representations required to build assembled object representations are modelled as class instances. But, unlike the TROPES system, it is based on the class / instance approach, and it is directed toward the engineering design process. And, unlike the SYSTALK system, and ROME language and the GOM model, OME tries to separate the information model of the product that supports the design process, and the system that aids to this design process. The goal is both to be able to use the OME model on any kind of CAE system, whether it is object oriented or not, and to allow the exchange between different CAE systems of multiple perspective information model of the product.

3. Abstraction mechanisms in engineering design

3.1. Instance and occurrence

The concept of technical object is the first abstraction mechanism used in engineering design. A technical object is an abstraction that allows to represent a set of (possibly unknown) physical objects considered, from a certain point of view, as interchangeable. We call such an abstraction a technical object *instance*. Technical objects may be defined at different *levels of abstraction* representing more or less wide sets of physical objects. An SKF 6200-ZNR bearing is a technical object that

represents a set of interchangeable *supplier parts*. An ISO 30 BC 02 XE bearing is a more abstract (i.e., representing a wider set) technical object, it may be called an *abstract part*. A sphero-cylindrical bearing is a more abstract technical object that may be dealt with during the kinematics analysis stage of a design process (it fixes only three degrees of freedom).

Each technical object is defined by some human organization (e.g., SKF, ISO...) who gives to it a *name*, that identifies the abstraction, and who specifies some (generally few) features that shall apply to all the physical objects represented by the instance.

When an instance is selected, and inserted in a design, some additional informations are linked with the instance that becomes an *occurrence*. Some of these informations, called the *insertion characteristics*, characterize the positioning of the occurrence in the CAE modeling structure (e.g., coordinate space, placement, set structuring, layer...). Some other informations, called *context parameters*, may characterize the elements of the context of insertion that modify the occurrence properties (e.g., the compressed length of a spring, the dynamic load of a bearing) and other ones, called *representation attributes*, the reaction of the occurrence to this context (e.g., distance between whorls of the spring, lifetime). A complementary *identification schema* is required, at the occurrence level, to carry these additional informations.

3.2. The concept of models

Even if they are used in the other domains of knowledge [MIN 86], the systematic and formal use of (technical) models is the main characteristic of the engineering and of the scientific process. An old definition from MINSKY, quoted in [MEL 67], directly leads to a multiple perspective point of view: "for an observer O, an object M is a model of an object A if M helps O to answer some questions he wonders about A."

The first point that results from this definition is that a model is a *dynamic object*. Providing answers to some set of related questions it shall encapsulate both specific data and dynamic operations. This feature precisely fits with the (object oriented) concept of object. The second point is that a model only exits as a structure that may provide answers to *some* questions about *some* objects. It may be *characterized* by the questions it supports, and by the objects of which it is an abstraction. The third point is that a model depends on the observer, or, from our analyst point of view, from the designer of the information model: very different models may answer to the same kind of questions (e.g., finite element models or abaci for stress analysis, geometric constraints or mathematics equations for kinematics, parametric programs or set of geometric entities for geometry), and very "different" questions may be supported by the same model. It is a non predefined designer choice depending on the problem domain. The last point is that a model preexists to any occurrence in any engineering design. It is an abstraction mechanism that stays at the instance level… and which is mainly used to provide answers about

occurrences of this instance. The simplest example is the one of geometric representation. The model of each technical object, wether it is system defined (e.g., a circle) or it is defined in some external library (e.g., a bearing) is basically a dynamic object (a program). Geometric representations of occurrences are *created* by this model, often as a set of data.

3.3. Models and views

The engineering design process always consists in selecting more or less abstract preexisting object instances, and putting them, as occurrences, inside the information model of the product under design. In a CAE system, the need for "representations" refers to both the instance and the occurence level. A major difference is that instances preexist to the design process. Their representation may (in fact: should) also be preexistant. Occurrences are dynamically created during the design process. Hence, the same shall apply to their representations.

The concept of model discussed in the previous section provides a straightforward approach for modelling instance representation. An instance representation, called a *model*, is a dynamic object, (i.e., a class instance), which:

– refers to a technical object instance,

– answers to some precise questions about this technical object instance.

Besides information retrieval, the main questions from the CAE system user are to create occurrence representations inside a product information model. Hence, the questions a model has to answer may be formalized by the name of the category of occurrence representation, and, possibly, the level of occurrence representation required by the user for some well-defined occurrence.

The results of this question, called (occurrence) *view*, generally has a very different structure from the model one. It refers to a technical object occurrence, it is caracterized by the question that it is an answer to, it is often only made up of data (with possibly a reference to the model that created it).

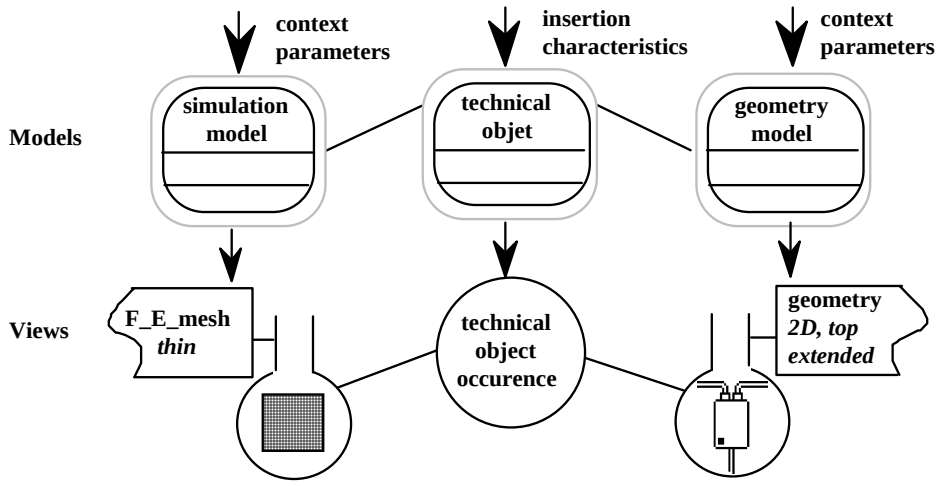

Fig. 1: *Models and views*

4. Multiple perspectives modelling of the engineering design world

The three kinds of abstractions discussed in the previous section: technical objects, models and views, are to be represented inside a CAE system. They are represented by different class hierarchies.

4.1. Modelling technical objects: general model classes

The technical objects are naturally apprehended through classes and class hierarchies. It is a basic method of organizing knowledge for human beings [MIN 86] [COA 91]. A major controversy in the object oriented community is about the unicity of the class hierarchy. Each time we adopt a different perspective about a set of classes, it suggests a different classification hierarchy. For instance, electronic components may be classified according to their functional characteristics, their technological characteristics, their manufacturing characteristics, and so on. We assert that this problem arises when we try to represent several representation categories in the same hierarchy: all the component catalogues, all the standards, all the technical books follow a simple tree structure. Modelling each perspective through a different simple hierarchy vanishes this problem from a generalization / specialization point of view. To distinguish the user access hierarchies from the *is_a* hierarchy (and to make the former customizable) vanishes this problem from the classification point of view.

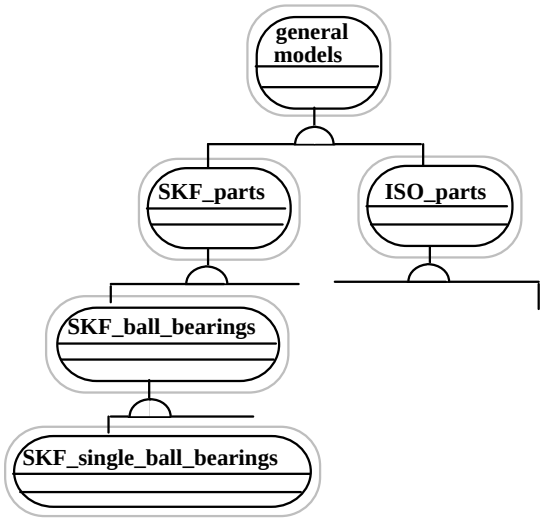

Fig. 2: *Simple hierarchy of General model classes*

We call *general_model classes* the classes which model technical objects. Technical objects being abstractions, these classes basically describe the identification of these abstractions, and, possibly, the fundamental and multiperspective properties associa-ted with them by the entity that defines it.

Constraints more complex than Cartesian product of type domains generally restrict the legal instances of classes [PIE 90]. These *integrity constraints* are to be defined, and ensured, at the class level. The *new* instanciation class method shall ensure that a created instance fulfills these constraints. Another class method, *change_object,* shall permit to modify an instance under the control of the class.

A general_model instance supports the two corresponding methods for occurrence representation. The *create_general_view* allows to create an occurrence of the technical object instance in the CAE modelling structure. The method requires the insertion characteristics of the occurrence and creates a representation, called

general_view, that carries the identity and the fundamental properties of the technical object as they are defined in its general_model class. The *change_general_view* creates an occurrence by re-using the insertion characteristics of a preexisting occurrence provided as parameter of the method.

The interpretation of the sub-superclass *is_a* relationship is the strict set inclusion. Hence, all the properties associated with a class (and in particular, the *is_view_of* relationship, see § 4.2) apply to any of its subclass.

Two kinds of non completely defined instances may be allowed by a CAE system. On the one hand, some non-terminal classes may be allowed for instanciation (e.g., a "bearing", instance of the superclass of several bearing classes). Such an instance is called a *generic* technical object. On the other hand, it may be possible to create terminal classes'instances of which some attributes have the predefined *null* value. This allows the CAE system user to define only those characteristics of the technical object that are relevant for the stage of its design.

4.2. Hierarchies of functional model classes

Parallel hierarchies of classes, called *functional_model classes*, permit to represent the various categories of technical models necessary in the problem domain. A class of functional_models:

– refers to one or several classes of *general_models* for which it provides technical models,

– declarates the attributes of an instance of a *general_model* class which shall exist and have a non null value in order, for this instance, to be allowed as value for the predefined *ref.* attribute,

– defines the (possible) *context parameters* whose values shall be provided to create the occurrence views it is able to create,

– defines the *representation attributes* which are attributes of its instances,

– contains the *derivation functions* which allow to derive representation attributes from the *general_model* class instance attributes, and, possibly, from the context parameters values (these functions may be defined through some static data structure defined within the class),

– describes the view creation *methods* supported by its own instances.

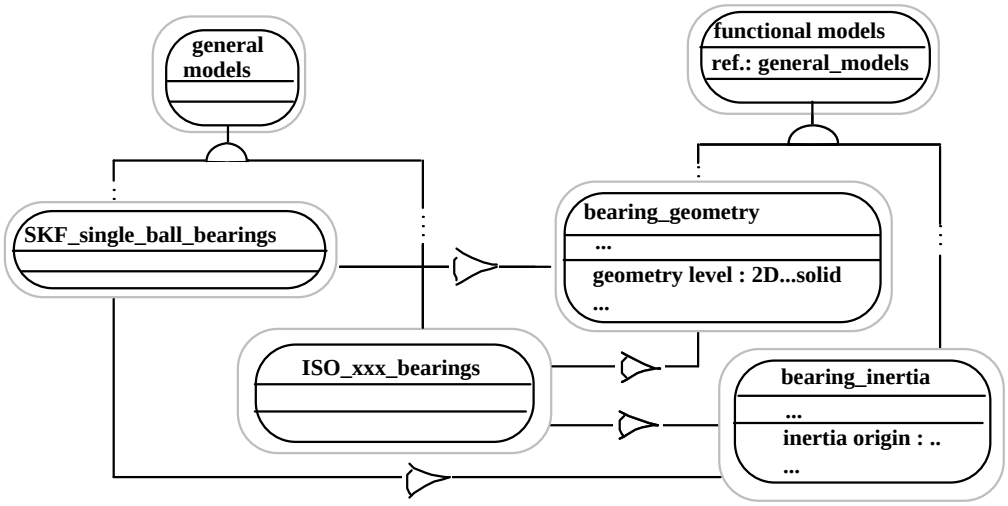

Fig. 3: *Parallel hierarchies of general model and functional model classes*
(⋗ represents the is_view_of relationship)

Although it is declarated at the class level, the relationship between a *general_model* (i.e., an instance of a general_model class), which represents a technical object, and a *functional_model* (i.e., an instance of a functional_model class), which represents a technical model of this object, is an object / object instance connection [COA 91]. It is called the *is_view_of* relationship. When a class is linked by the *is_view_of* relationship to another class, this means that there exists one legal instance of the former corresponding to each instance of the latter.

The role of a functional_model is to create various representations, called *functional_views*, of occurrences of the general_model it is *view_of*. A functional_model contains representation attributes that represent the data needed to create some category of functional_views. It supports methods that create these functional_views.

The methods of functional models are activated through message passing. These messages are called *view creation message*. The selector of a view creation message is the name of the representation category required. The possible parameters describe the level of representation. The result of the method is to create a functional view of the occurrence of the instance of which the functional model is *view_of*. The different hierarchies are independent, and the *is_view_of* relationship is inherited through the *is_a* hierarchy.

When a general model receives a view creation message, it generates a *search_for* message. The system follows the *is_view_of / is_a* lattice and searches for a class that supports the view creation message. It sends a *new* message to this class, then it sends the view creation message to the resulting instance.

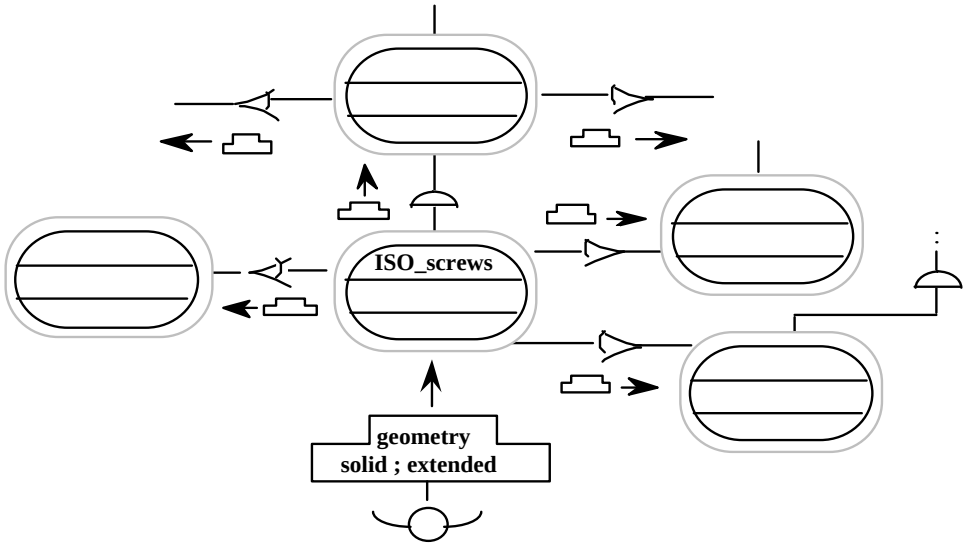

Fig. 4: *Message passing for view creation*

For instance, a class of geometry functional model that is *view_of* a class of springs contains:

– the attributes which describe all the geometric characteristics of this spring class;

– the *derivation functions* which describe the functional dependencies between these attributes and the attributes defined in the general model class of springs ;

– the context parameter (compressed_length) type and integrity constraints ;

– the parametric program(s) which allow(s) to create one or several levels of geometric representation of the various springs of the general_model class.

4.3. Hierarchy of view classes

The views are the occurrence representations that are created inside the information model of the CAE system. Hence, the structure of each category and level of views shall be independent of the technical object that the view refers to, and shall be known by the CAE system. The different categories of views are defined by a third class hierarchy. The specific class *general_view* describes the information model of technical object occurrence. The other views (functional_views) describe the information model of each category of views. The values of view attributes are produced by functional_model methods. According to the CAE system structure, they may be accessed by the CAE modelling software either through message passing, or directly.

4.4. Whole-Part Structure

The aggregation relationship *is_part_of* permits to describe an assembly that is made up of simpler technical objects (e.g., a bolt = a screw + a nut) which may be generic. The general_model class that defines the assembly describes the integrity constraints (e.g., the_bolt.thread_diameter = the_nut.thread_diameter). The

functional model classes, which are *view_of* the assembly class, create functional views of the assembly through message passing to the general model of the assembly parts. Such a view may be either an atomic functional view that refers to the assembly occurrence, as it is the case in the CAD-LIB standard [PIE 90]. It may also be a set of functional views, one for each part of the assembly and one more which describes the functional relationship between the view of the parts, like in the SYSTALK system [WOL 91].

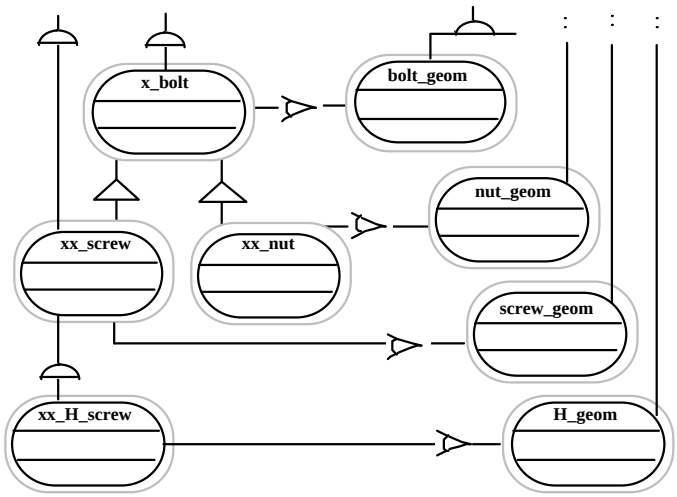

Fig. 5: *Modelling assemblies*

5. Implementation issues

The model, presented in this paper, has been designed to allow for portability of libraries of technical objects on different kinds of CAE system. The first version of this model [PIE 88], has been extensively studied and refined in a project involving three CAD system suppliers, and one software house. Four prototypes were implemented in this project [PIE 92] and are able to operate on the same library. One prototype was based on both a general purpose object oriented (OO) database management system (DBMS) and an OO CAD system (MTEL® from Caroline Informatique), the second one was based on a general purpose OO DBMS (ECCI® from IDEC) and a non OO CAD system (STRIM 100® from CISIGRAPH), the two last ones were based on specific DBMS and on non OO CAD systems (EUCLID® from MatraDatavision and DMT® from MECASOFT). A fifth implementation, which will become a product, is based on a relational DBMS and uses an object layer similar to PENGUIN [BAR 90]. The diversity of these implementations demonstrates the feasibility of the implementation of the proposed model in very different environments.

At the standardization level, this model makes up the basis of the European CAD-LIB standard (pr ENV 40 004), and of the work of ISO/TC184/SC4/ WG2. The goal of this later is to allow for portability of complex libraries of objects in connection with STEP. The hierarchies of general_model classes and of functional_model classes that constitute a supplier library are described in the formal language EXPRESS (ISO DIS 10303-11). This allows the use of the STEP physical file

structure (ISO DIS 10303-21) to exchange complex libraries. The different parts that compose these Standards have been approved by their respective standardization committee.

Conclusion

A major challenge of the CAE technology is to be able to integrate the various "islands of automation", each one with its own representation, which already exist in the engineering design process. Such an integration requires a robust model able to support multi-representation of the product components, and which may be implemented in various environments. Object Oriented technology provides a straightforward approach for modelling the concept of "technical model" that is a major characteristic of the engineering process, but it does not provide adequate tools to deal with different perspectives.

The information items involved in the engineering design process may be captured at three levels of abstraction: the class level, the instance level and the occurrence level. A multi-representation problem exists at each one of these three levels. The *is_view_of* relationship presented in this paper proposes a solution for these three problems. It permits a distinction between what a product *is*, which may only be defined by *names*, and some fundamental *predefined properties* (e.g., positioning), and what a product *has*: an unlimited number of representations. Hence it provides, at each level, a root to which all the representations may be connected. Initially designed to model and exchange libraries that contain classes of multi-represented technical objects, the *is_view_of* relationship defines a mechanism, which may also be used, at the occurrence level, to improve the multi-representation capabilities of the present CAE systems, whether they are object-oriented, or not.

REFERENCE

[ATK 89] M. ATKINSON, D. DEWITTE, D. MAIER, F. BANCILHON, K. DITTRICH and S. ZDONIK: "The Objet-Oriented database system manifesto", Proc. of the First International conference on Deductive and Object-Oriented Databases, W. Kin, J.M. Nicolas, S. Nishio, Eds., Kyoto Research Park, Kyoto 1989, pp 40-57.

[BAR 90] T. BARSALOU, G. WIEDERHOLD: "Complex objects for relational databases", Computer Aided Design, 22, 8, 1990, pp. 458-468.

[BEE 89] E. BEEKER: "Application de la programmation orientée objet à la CAO", Proc. of MICAD'89, Hermes, Paris, 1989, pp. 121-132.

[BEZ 92] P. BEZIAT: "Assemblages et composants mécaniques", Proc. of MICAD'92, Hermes, Paris 1992, pp. 615-625.

[BOB 77] D.G. BOBROW, T. WINOGRAD: "An overview of KRL, a Knowledge Representation Language", Cognitive Science, 1, 1, 1977.

[CAR 88] B. CARRE, G. COMYN: "On multiple classification, point of view and object evolution", In: *Artificial Intelligence and Cognitive Science,* J. Demongeot, T. Herve, V. Rialle, C. Roche, Eds., Manchester University Press, Manchester, UK, 1988.

[CAR 90] B. CARRE, J.M. GEIB: "The point of view notion for multiple inheritance", Proc. of ECOOP / OOPSLA'90, October 21-25, 1990, pp. 312.

[COA 91] P. COAD, E. YOURDON: *Object-Oriented Analysis*, Prentice-Hall, Englewood Cliffs, N.J., 1992.

[DEM 78] T. DEMARCO: *Structured Analysis and System Specification*, Prentice Hall, 1988.

[FLA 92] J.F. FLANIGAN: "Business Issues Concerning Electronic Data Books", Proc. of the Test and Design Expo Conference, May 1992, USA.

[GAN 79] C. GANE, T. SARSON: *Structured Systems Analysis: Tools and Techniques*, Prentice Hall, 1979.

[GOL 83] A. GOLBERG, D. ROBSON, *Smalltalk 80: The language and its implementation*, Addison Wesley, 1983.

[GUI 89] J. GUILLOT, J.Y. ROUSSELOT, J.C. VIGNAT: "Conception assistée par ordinateur d'ensembles mécaniques avec recherche d'une bonne solution: le logiciel: SICAM", Proc. of MICAD'89, Paris, Hermès, 1989, pp. 197-215.

[KIM 90] W. KIM: "Object-Oriented databases: definition and research directions", IEEE Trans. on Knowledge and Data engineering, Vol. 2, n° 3, september 1990, pp. 327-341.

[MAR 90] O. MARINO, F. RECHENMANN, P. UVIETTA,: "Multiple Perspectives and Classification Mechanism in Object-Oriented Representation", Proc. of ECAI Conf. Stockolm, July 1990, pp. 425-430.

[MEL 67] J. MELESE: *La pratique de la Recherche Opérationnelle,* Dunod, Paris, 1967.

[MEY 88] B. MEYER: *Object-Oriented Software Construction*, Prentice Hall, Hemel Hempstead, UK, 1988.

[MIN 86] M. MINSKY: *The Society of Mind*, Simon & Schuster Inc., New York, 1986.

[MOR 86] J.M. MORANNE: "La CFAO en Construction mécanique, Bibliothèque de standards et cadre d'application, la démarche du groupe St Gobain", Proc. of MICAD'86, Hermès, Paris 1986, pp. 41-71.

[PER 92] J.F. PERROT, F. WOLINSKI: "Modélisation par objets en robotique", Technique et Science Informatique, 11, 1, 1992, pp. 97-115.

[PIE 88] G. PIERRA: "Proposition de modèle de référence pour une bibliothèque de composants standard multifournisseurs et multifonctions", Actes du Colloque Bibliothèque de composants pour la CAO et applications, AFNOR, Paris, 6-7 Sept 1988, pp. 4-30 (English transl.: CEN / CENELEC / CAD-LIB / N.24, Sept 1988).

[PIE 90] G. PIERRA: "An object oriented approach to ensure portability of CAD standard parts libraries", Proc. of Eurographics'90, Montreux, Sept. 1990, pp. 205-214.

[PIE 92] G. PIERRA: "Report on the contract N 90.1.3134" French Ministery of Industry, Paris, May 1992.

[SCH 77] K. SCHOMAN, D.T. ROSS: "Structured analysis for requirements definition", IEEE Trans. Software Engineering, SE-3, 1, pp. 6-15.

[STR 86] B. STROUSTRUP: *The C++ Programming Language,* Addison Wesley, 1986.

[SYV 92] G. SYVERTSEN, F. LILLEHAGEN, M. LOVSTAD: "A generic object model for engineering design", Proc. of TOOLS EUROPE'92, Dortmund, March 30-April 2, 1992.

[TOU 92] P. TOURON, D. BRUNIER-COULIN, T. NGUYEN: "CAS. CADE, un environnement logiciel pour bâtir des applications CFAO", Proc. of MICAD'92, Hermes, 1992, pp. 379-391.

[WOL 91] F. WOLINSKI, J.F. PERROT: "Representation of Complex Objects: Multiple Facets with Part-Whole Hierarchies", ECOOP'91, Genève, pp 288-306.