



Science Arts & Métiers (SAM)

is an open access repository that collects the work of Arts et Métiers Institute of Technology researchers and makes it freely available over the web where possible.

This is an author-deposited version published in: <https://sam.ensam.eu>
Handle ID: <http://hdl.handle.net/10985/16788>

To cite this version :

Serge TICHKIEWITCH, Lionel ROUCOULES - CoDE: a Co-operative Design Environment. A new generation of CAD systems - Concurrent Engineering: Research and Applications - Vol. 8, n°4, p.263-280 - 2000

Any correspondence concerning this service should be sent to the repository

Administrator : scienceouverte@ensam.eu



CoDE: a Co-operative Design Environment.

A new generation of CAD systems

ROUCOULES Lionel, TICHKIEWITCH Serge
*Laboratoire Sols, Solides, Structures de Grenoble -France-
Domaine Universitaire BP 53
38 041 Grenoble Cedex 9
France
Tel : (+33) (0)4 76 82 51 44
e-mail: Lionel.Roucoules*

ABSTRACT

According to conceptual changes on design methodology, this paper presents some research results on a new generation of CAD systems. This new software has not to be used to achieve automatic design tasks at all. On the contrary, it has to make part of an integrated design environment.

A co-operative design modeller is then presented as a support for co-operative and integrated design methodology in this design environment. It mainly allows every design actors to share a unique database owing to a formal exchange network. The modeller provides a graphic interface to add, edit or modify data. Moreover the modeller manages the shared database in order to realise heavy design tasks as data propagation or data coherence management.

The presented modeller takes place in the global design environment connected to specific applications based on specific design tasks (process planning, structure analysis). An informal communication network worthwhile in any design group including human actors also assists the co-operative design modeller.

KEYWORDS

Integrated design, co-operative design, CAD system, Product Model, Multi-view, product emergence.

INTRODUCTION

The evolution of CAD systems in mechanical engineering during the eighties and nineties has made industries been more and more competitive in design.

During the eighties, design modellers were called CAD system but only assisted usual drawing operations. They made easier the creation of form features, their modifications and their safeguards. In this way, engineers were faster and faster in drawing systems and parts but those systems were not really CAD systems. Later researches during the nineties tried to integrate trade knowledge in those drawing modellers. The main objective was then to calculate form dimensions taking into account trade constraints as drawing rules. We had automatic design tasks and self-learning, we thought about artificial intelligence. Step by step, more data than form data were taken into account and were calculated following the state of the product design. Those new systems called « expert systems » were quickly limited in their utilisation during the design process and further, during a co-operative design. Indeed, they were very specific and could only be used in specific design as tooling [1], stamping [2], asynchronous electrical motor design [3]...

Nowadays, integrated design objective is to take into account a maximum of knowledge during the design process [4]. This knowledge comes from all trade actors of product life cycle. Moreover, this design methodology allows the simultaneous realisation of design tasks [5] gathering all actors to discuss on the design project.

From this new industrial organisation, several researches try to develop new CAD systems. This new software does not throw away expert systems but has to assist the connections of each ones.

First of all connections have to provide a large pool of data that can be shared by several design actors. These data can be relative to a specific trade as structure analysis for example and be connected to form data [6]. Second of all, integrated design requires the software to be used as soon as possible during design. Such conditions are supplied using functional specifications to choose specific technologies. The technologies are both structural technology [7] [8] and trade technologies as structure analysis for example [9]. Thus, we break the barrier between product functions and product structure [10].

Despite of those results, CAD systems are not a real support for co-operative design. Such support must manage different kinds of data and assist the co-operation among designers. It then assists the progressive definition and constraint, the design arrangement and the emergence of the product. One part of our research concerned by these problems is presented in this paper.

The global presentation of an integrated design environment is done in part I of the paper and is based on previous references. This environment is mainly composed of an integrated design modeller, of an informal exchange network and of several specific trade applications. These applications are those introduced just above as the evolution of the expert systems. The environment allows the integration of each trade actors on the design project adding their data and their constraints and using their own specific trade software.

Part II deals with the computer developments of the integrated design modeller as part of the global integrated design environment.

Developments are first constrained by the used computer environment. We use C++ object classes and several object libraries provided by Ilog society (Views, Server, Broker and Solver). Ilog libraries are coupled to Motif and Inventor libraries from Silicon Graphic. Developments are also constrained by the limits of the modeller in use. New functionalities are therefore progressively added.

The modeller realises two groups of tasks: the introduction support and the management of data. The data introduction support is made owning to a graphic interface that allows each actor to add, edit or modify data on the design project. These manipulations are based on the association between a knowledge model (the features) and a data model and on the multi-representations of each data. A data management task consists in structuring data with specific data model (components, links, and relations...). This management keeps the coherence in the data model and between the data in order to assist heavy tasks and to show design conflicts.

Finally part III details a design example. This example, using the integrated design modeller, shows the progressive emergence of product data. Functional surfaces and every data come from the choice of different technologies. First Technologist actors impose the structural technology for the main functions of the product. Afterwards, other actors choose specific technologies (tooling, stamping...) and thereby add new data and new constraints.

I. PRELIMINARY CONCEPTS

Integrated Design is one of the thematic of the Soils, Solids and Structures laboratory. The results are both conceptual and practical. On the one hand, they are working on design methodology. They study how each actor is able to reach the design project as soon as possible in the conceptual or detailed design stage. On the other hand some socio-technical studies of industrial design process bring up information in order to develop new CAD systems. This paper gives some results of their developments on an integrated and co-operative design environment. This environment allows the collaborative participation of several actors on the design project. These actors are therefore all connected to a shared database through four-access level as it is shown on Figure 1. Main concepts on the design environment presented in [11] are summarised in this part and introduce the computer developments detailed in part II.

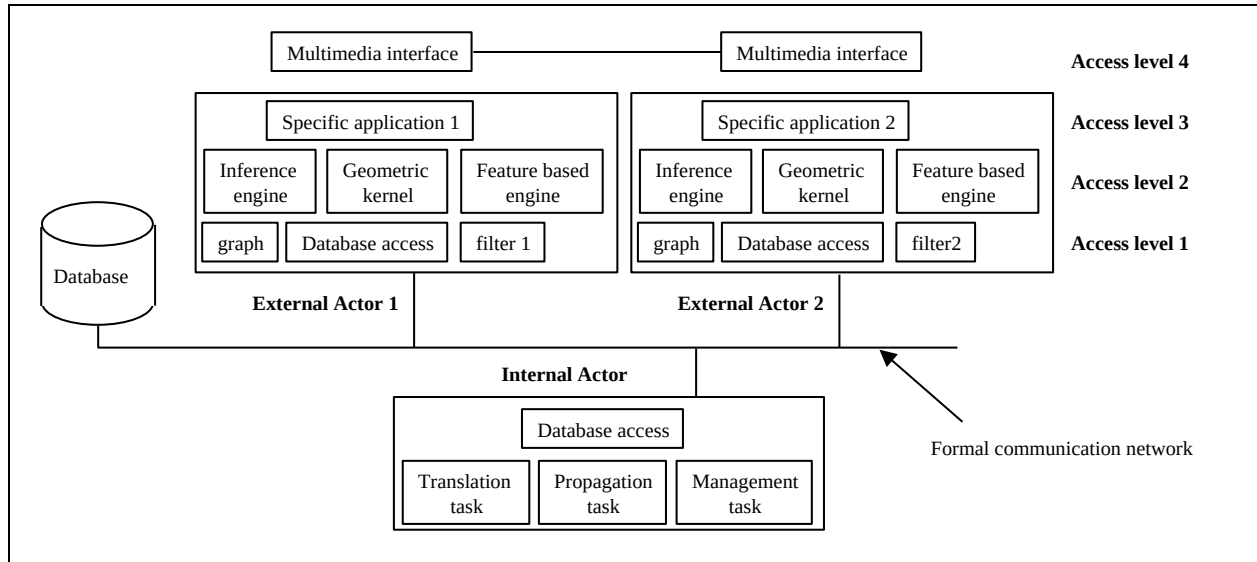


Figure 1: Conceptual organisation of a co-operative design environment.

1. Internal, external actors and knowledge concepts

In co-operative design, several actors participate to the design at the same time. In this situation an actor has to realise some tasks during the design process. One can map the functions to the structures whereas other ones participate to the structure analysis or to the manufacturing definition...

External actors are physical persons in the design environment that have a special point of view on the product. They bring their own knowledge to have the design evolved. A knowledge is thus related to a design trade as tooling, stamping... Nevertheless one actor can have several trade knowledge. In a lot of industrial situations, stamping and tooling constraints are so closed that the same person can own them.

Internal actor realises tasks but it is not a physical actor. The internal process of the environment manages these tasks. As shown in part II, such tasks are automatic and transparent for external actors. In other words, they make the dirty work of the design (data translation, data propagation, data coherence management...).

2. Co-operative Design Environment (CoDE)

The computer development of CoDE is shared in three parts that include the four access levels and the shared data base previously presented. Each part presented on the Figure 2 will be here briefly presented before the presentation of the Co-operative Design Modeller (CoDeMo).

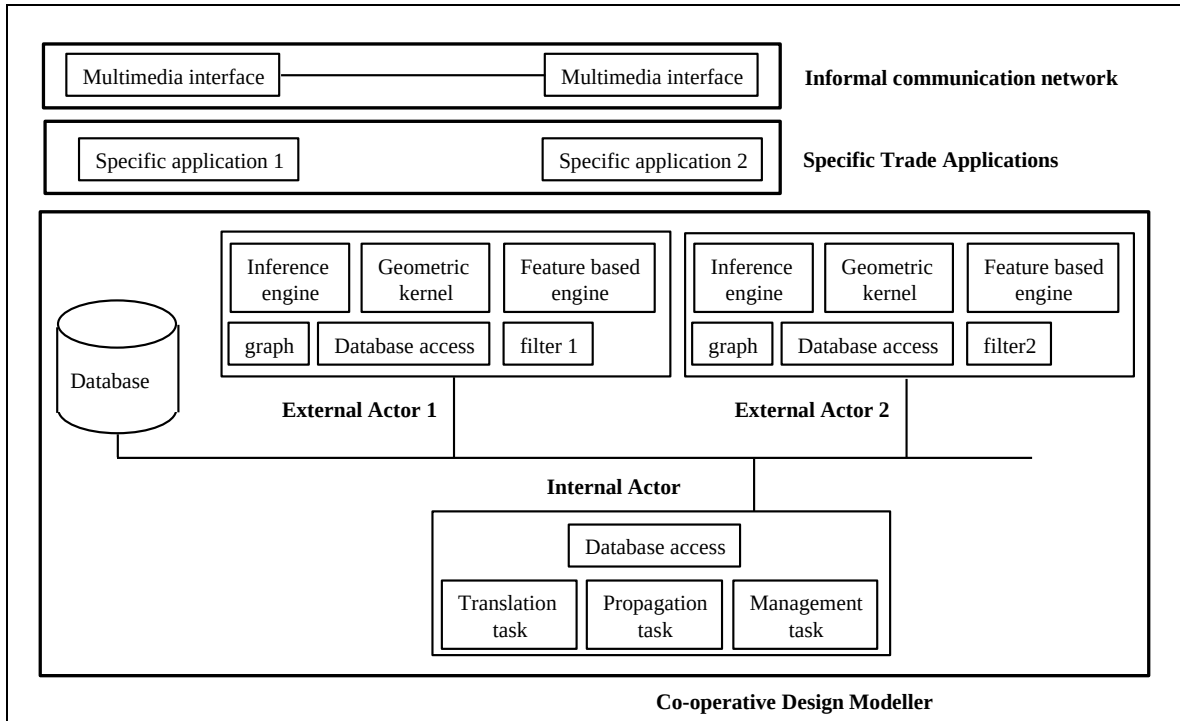


Figure 2: Different parts of CoDE related to the initial concepts.

2.1. The Co-operative Design Modeller (CoDeMo)

As we said in the introduction the real tools for actors to co-operate and to share data during the design process are the main lack in CAD systems. CoDeMo objective is not to integrate knowledge in order to take automatic design decisions as it was with the development of expert systems. CoDeMo assists the integration and the co-operation of each participant who would be external or internal actor. It achieves thereby a formal network of co-operation based on the shared database, which is manipulated by every design external actors and managed by the internal actor.

External actors use a graphic interface in order to have specific views (points of view) on the product. From these views and through the formal network, they modify or create new product data, which is added to the shared database. In such case, new data are notified by CoDeMo and therefore seen by every actor.

Internal actor manages the database structured with a data model proposed by MatraDataVision in 1994 and based on Components, Links and Relations objects [12]. Chapa added to this model the multi-view concept [11]. This last amelioration takes into account the large kinds of knowledge used during the design. It then allows the decomposition of the product in many groups of data related to specific trade knowledge. Figure 3 shows different views of the decomposition of a thermal motor using the component, links and relation data model and its basic representations.

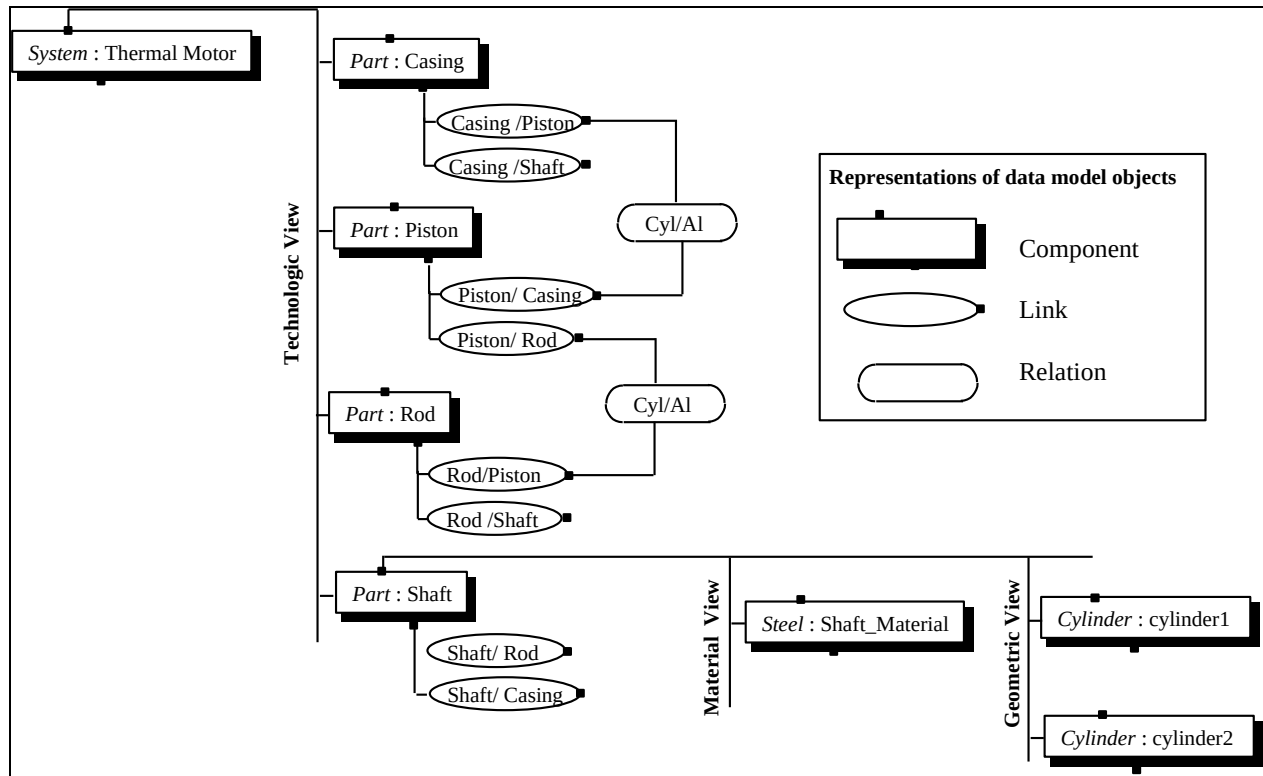


Figure 3: Partial graph of a thermal motor multi-view decomposition using Component, Links and Relations.

2.2. The specific Trade Applications

The *specific trade applications* used in CoDE are actually the new generation of expert system. These applications are developed in a specific trade context to realise precise tasks. They allow specific evaluations of the product with different points of view. Such new applications are developed to be used in an integrated design methodology and not any more as knowledge manager and auto-learning software as they were during the artificial intelligence period. For example, [13] presents a CAPP system that defines the tooling process planning of the product according to the functional surfaces and tolerances.

First, *specific trade applications* have to collect minimal data needed to run their own process. For instance in most of CAPP software, the system needs the geometry, the functional tolerances and the material of the product. It calculates afterwards tooled surfaces as well as tooling data like feed or cutting speed...

These data are then introduced in CoDeMo to progressively define and constraint one solution. This operation is still manual and thereby obliges each actor to define his real description, his real point of view on the product. Obviously a formal connection using exchange file could be useful and would make easier the job for external actors. It will so be necessary to study the most appropriate standard exchange format.

2.3. The informal communication network

The formal network (the shared database and the internal actor) actually manages the data coherence. It thereby detects automatically design conflicts. These conflicts are the sign of incoherence between several constraints brought by the actors and have to be resolved by actor arrangements. In this situation an informal network seems interesting. It has to be seen as a normal mean of communication assisting the traditional dialogues of a design group or between different departments of the industry.

Normal dialogue means that the informal network has to permit writing, talking, sketching and watching each other in order to discuss about a problematic data, a part or an assembly of the product. Nowadays, such kind of products like *Microsoft NetMeeting* is already on the computer market. It will be interesting to study their insertion in CoDE.

3. Content improvements and discussion

So far, the paper has just presented what were the capacities and the functionalities of CoDE. Nevertheless it is interesting to describe the real improvements of such a design environment discussing them with what is already existing or what is currently investigated.

3.1. *The shared database and the multi-views decomposition*

Concurrent Engineering as presented in [Sohlenius 92] aims at shorting product design time. The reduction can be made during product definition but also during order to the production.

One solution is to realize simultaneous tasks as far as possible. Recently, Aldenondo has presented a system architecture called *Caméléon* that can be adapted to specific tasks to achieve them concurrently instead of one after the other [Aldanondo et al. 00].

Another way to accelerate the product definition is to avoid design iterations among each department that normally collaborates on the design. Instead of pushing the information from department to department, the idea is to pull it from a shared database where data are well structured [Lutters et al. 97]. Thus, every department can reach the information whenever it wants (as soon as possible), and works on upgraded data.

Such a shared database is used in CoDE and is structured using a specific data model as previously presented (cf. 2.1). The multi-views concept allows the integration of every kind of product data relative to the product life cycle. It emphasizes the integration of every actor of design as soon as possible. This kind of information is not taken into account in current modellers on the market (*SolidWorks*, *ProEngineer*) that focalise the product definition on a geometric model.

3.2. *A client/server application based on RPC connections*

As far as we talk about Concurrent Engineering or Integrated Design, a lot of designers have to work together. Nowadays industries gather those designers in order to participate to the product definition. The advantage provided by CoDE is to be able to connect the shared database via a formal network. Designers have not to be gathered any more to work all together on the same project.

The formal network is based on a RPC (Remote Procedure Call) architecture between a *client* and a *server* processes. Those two processes are detailed in part II of the paper.

Other architectures are now developed in order to realize distributed application: CORBA (Common Object Request Broker Architecture), JRMI (Java Remote Method Invocation). For users, all those architectures are similar. Indeed the difference remains actually in programming the IDL (Interface Definition Language) that is used to generate stubs for the client and the server.

In our case IlogBroker libraries are used. These libraries define a higher layer for distributed applications. The programming is easier owing to an IlogBroker pre-processor that create the stubs according to your personal C++ classes definition. IlogBroker pre-processor can currently provide both RPC and CORBA architecture.

3.3. *Compare CoDE and HTML technology*

In the literature some research look for web-based system for concurrent engineering [Huang et al. 99]. The HTML programming is interesting to use new technology for creating a graphic interface as we do with IlogViews libraries in CoDE. It is also an interesting technology to be used as a client / server architecture.

In fact, the use of Ilog libraries has been chosen to keep CoDE homogeneous from a C++ programming point of view. Indeed, It is based on several kind of libraries to create a GUI, to manage the data, to propagate the constraint and to achieve the client / server connections.

Nowadays, with the evolution of the HTML technology, it would be interesting to make tries in comparing different solution in term of information transfer speed and data propagation. The data propagation in CoDE is realised with notifications functions as explained in part II/2.1.

3.4. *A functional surfaces based design modeller*

The last conceptual improvement of CoDE remains in the design approach. Part III of the paper explains how the design modeller is based on functional surfaces provided by progressive knowledge integration. The product is not designed from a geometric model any longer, as it is the case in the actual CAD systems.

II. CoDeMo: A COOPERATIVE DESIGN MODELLER

In this part of the paper the Co-operative Design Modeller (CoDeMo) previously introduced in the whole context of the environment CoDE is presented. We present the current state of the system and the computer developments related to the concepts presented in part I.

All developments have been realised in C++ language using object libraries and headers files. *Ilog* products libraries Views, Server, Broker, Solver [18] are used and we detail the facilities provided by each product. Moreover, in order to realise the 3D graphic representation of data, *Motif* and *Inventor* libraries from Silicon Graphics are used. Both kinds of libraries are running together using a *Motif* event loop.

CoDeMo is mainly a client/server process; each one works on different tasks:

- the *client process* is used by external actor to manipulate product data through a Graphic User Interface (GUI),
- the *server process*, in other words the internal actor, manages the shared database as presented later in part II.2.

1. Data manipulation

Data manipulation means in fact how each actor can edit the already existing data in the database, modify the existing data and add new data on the product. They are thereby using the first level of database access shown on figure 1. In this way, we provide a graphic interface that supplies:

- the database access,
- the multi-view and multi-representations of data,
- the connections to specific trade applications.

1.1. Database access: Creation, Edition and Modification

The design of a product is a progressive definition and constraint of its data. Such data are in part geometrical data, which represent the form characteristics of the product, however data have also to define other characteristics as the materials, the mechanical properties...

During the design process, every designer treats information on the product with his knowledge in order to create new data. These data are formalised as features described by their characteristics and behaviours [19]. Features actually define a knowledge model and each ones are specific to a design point of view. We can separate two kinds of features used in the integrated design methodology:

- *descriptive features* that describe the product according to a specific vocabulary and therefore according to a specific point of view. For example, a *Cylinder* feature describes a cylindrical form of the product with its characteristics as the radius, the length, the area... A behaviour of this feature could precise that the area value is linked to the radius and length values,
- *constraint features* that are used to define constraint on descriptive feature characteristics. *Equality* is for instance a constraint feature. This one is defined with two characteristics: variable1 and variable2. Its behaviour imposes the two characteristics to be equal.

This knowledge model is then associated to the previously introduced data model (components, links, and relations). It is really the association of both models that contextualises design data (cf. Figure 4). The association is realised through the *Creation* operation and can afterwards be watched with the *Edition* operation and modified using the *Modification* operation. The result of the association is called *Product Model*, which contextualises the features associated to a specific product decomposition.

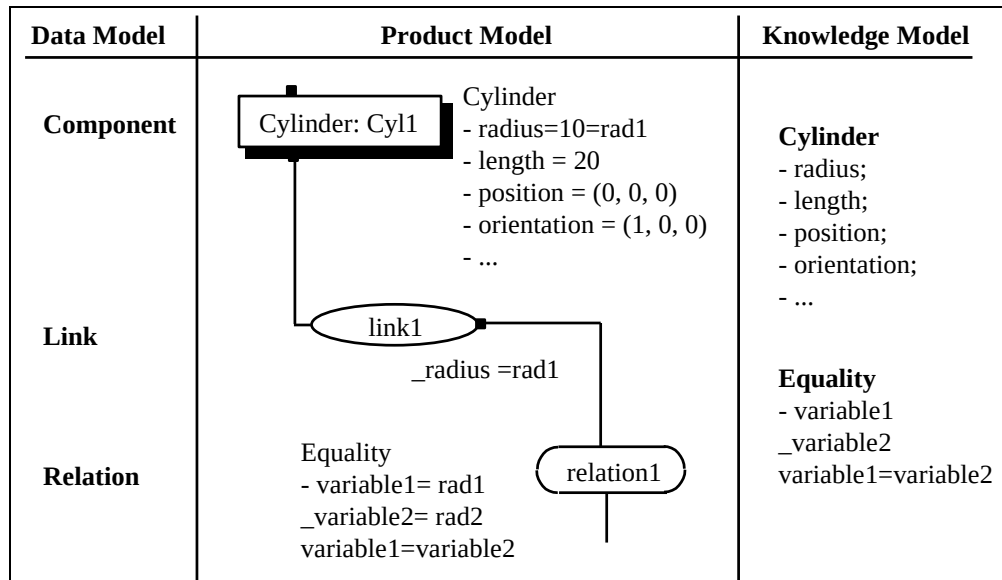


Figure 4: Association between the Data Model and the Knowledge Model to create the Product Model.

The **Creation operation** is done from the GUI of CoDeMo client process. Such operation can be explained in three parts (cf. Figure 5).

First, designer creates components as a granular description of the product. He starts the multi-view decomposition from the first component, which is associated to a *System* feature. This first component represents in fact the main project in the Technologist view. The actor has to choice a descriptive feature that precises the description of the product. Each feature belongs to a specific view and describes different granular level of the system, a subassembly or a part. Finally he creates a new component from the first or whichever one associating the chosen feature for example a *Cylinder* feature.

Secondly, the designer creates relations between product data. In this way, he associates parts of the knowledge model to links and relations of the data model. Links are created from a component and represent the possibility to hang this component. The actor therefore associates to a link a list of characteristics of the descriptive feature associated to the component. For instance, if a *Cylinder* feature is associated to a component, a link can own the radius characteristics. It means that the component can be hanged by the radius of the *Cylinder* feature. Once the creation of the links, the designer creates the relations. Indeed, a link is created because of the needed relation between several feature characteristics.

Thirdly, to end the construction of the product model, the designer associates a constraint feature to every relation. Such association describes a specific constraint between every characteristic associated to the links of the relation. For example, if two components associated to a *Cylinder* feature have each one a link associated to the radius characteristic, it could be useful to keep the equality between both radii. Design actor therefore associates an *Equality* constraint feature to the relation between both radii. Equality characteristics (variable1 and variable2) are in this case the two radii and the behaviour (variable1=variable2) imposes them to be equal.

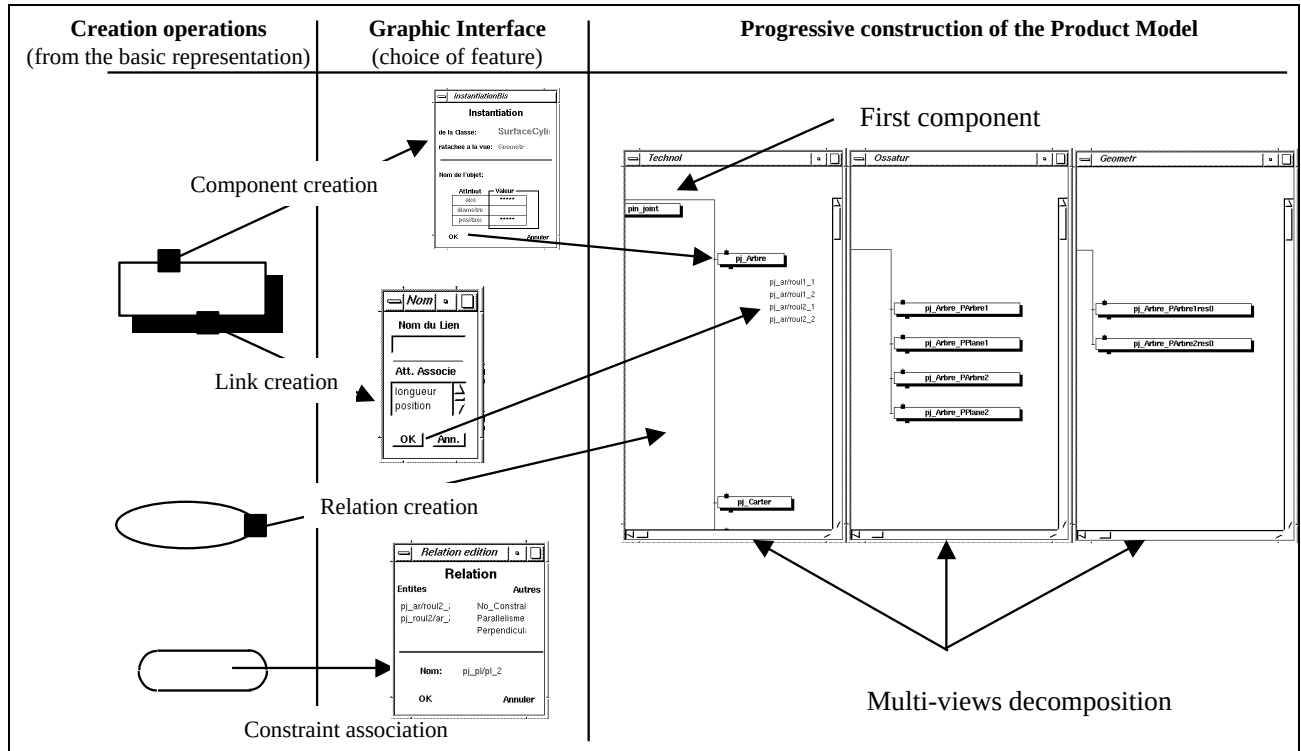


Figure 5: Creation operations for the Product Model construction.

The **Edition operation** is used to watch the knowledge associated to the data model. The GUI allows each actor connected on the database to edit the descriptive feature associated to a component, the characteristics associated to a link or the constraint feature associated to a relation (cf. Figure 6).

After the Edition operation, designers have the possibility to modify data in the Product Model. The **Modification operation** is just done using the facilities of the GUI that make the internal actor run modification procedures (cf. Figure 6).

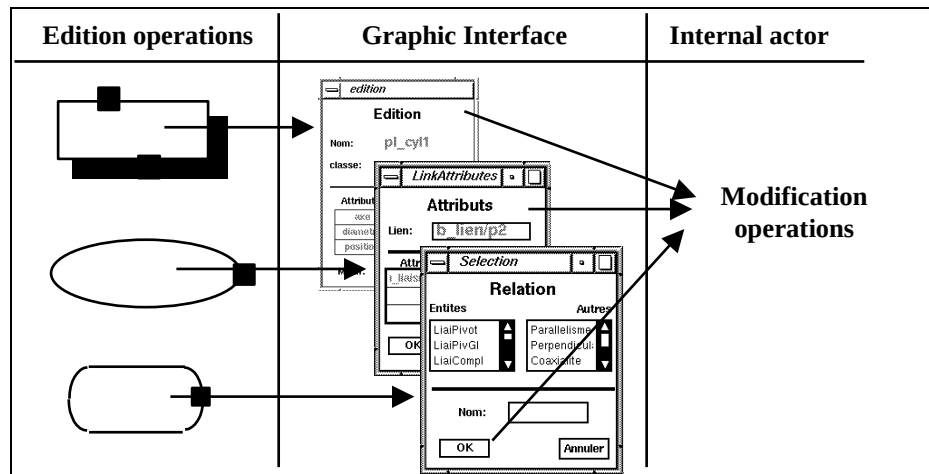


Figure 6: Edition and Modification operations in CoDeMo.

1.2. Multi-view and Multi-representations

We have just described how product data are progressively defined and stored in the database. Therefore the product has several decompositions according to different points of view; the product has a multi-view decomposition. The

multi-view decomposition was introduced by Chapa to gather every data that describe the product with a specific vision of it. We have mainly two kinds of views:

- the trade views (technologist, tooling, stamping, structure analysis...),
- the Ossature and Geometric views.

Trade views store the data that describe the product with a specific trade vision. For example, process-planning data make part of Tooling view of the product. From initial information, specific trades can create new data and can introduce them in specific trade view. We will explain in part III that those new data can define new functional surfaces introduced in the Ossature view, new description of the product in a specific view, or new constraints on already existing data.

Geometric and Ossature views of the product describe respectively the nominal dimensions (radius, length...) and the functional characteristics (roughness, tolerancing...) of every surfaces of the product. These views are not related to trade constraints (such trade does not exist) and have to be the results of translations from trade data.

Figure 7 details an example of the multi-view concepts using different descriptive features. A shaft, as a technologic part, can be viewed as *Cylindrical* skin and its roughness Ra3.2 in the Ossature view, as a *Cylinder* and its diameter D= 10 in the Geometric view and as a *Sliding* feature in the Tooling view.

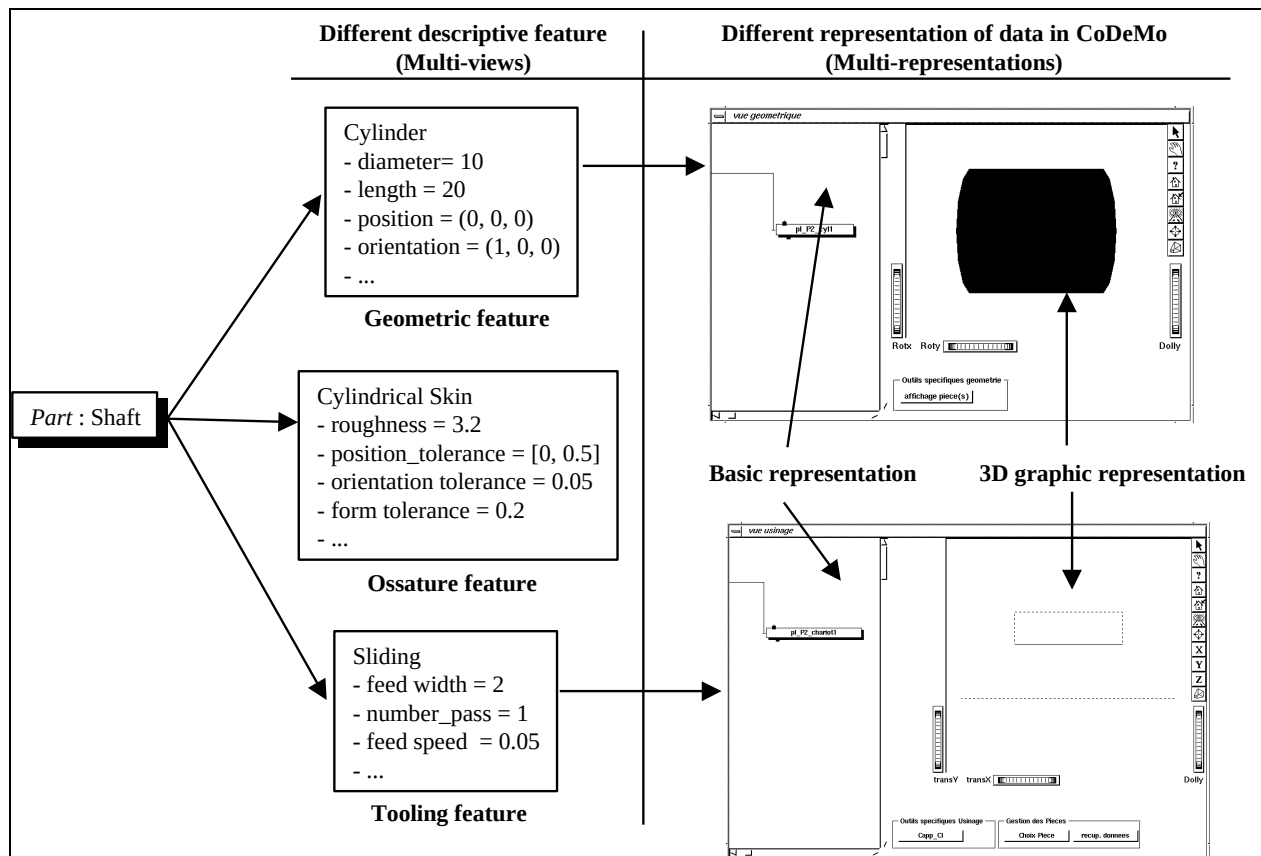


Figure 7: Different descriptions of the same object using specific feature and different representations in CoDeMo views.

A first kind of representation of data, further called *basic* representation, has been developed in CoDeMo. It is worthwhile for the manipulation of the database by external actors and looks like the graphic representation of the data model presented by MatraDatavision (cf. Figure 3). However such representation is not understandable. It just represents a tree of elements. By consequence, rather than a unique kind of representation, we decided to develop several representations. These new representations have to look like the usual representations used by trade actors. In this way they improve the comprehension of data in CoDeMo.

We first added a *3D graphic* representation that is currently used by most of designers. Figure 7 shows the two representations (basic and 3D graphic) of the shaft decompositions in the Geometric view. Nevertheless, the 3D graphic does not represent only form feature but every kind of features that has a usual 3D graphic representation. Tooling features for instance have such kind of representation (cf. Figure 7).

Despite of both basic and 3D graphic representations, it will be a good work to research the other kinds of representation needed in design. We can imagine for example a matrix representation for the Material view.

The computer development of the GUI has been realised using IlogViews and Inventor libraries. The first ones provide first predefined environment as windows, container, etc., to create the global structure of CoDeMo views. Then we can add a lot of gadgets (scrolled views, buttons...) useful for the dynamic of the views. IlogViews libraries also provide 2D graphics used to make the *basic* representation (rectangle, ellipse, round rectangle, line...). Next, Inventor libraries are used to make the *3D graphic* representation. These libraries predefine graphic viewer and 3D objects as sphere, cylinder...

Using both kinds of libraries, the main problem is to be able to manage both event loops: Ilog and Inventor ones. The solution remains in the integration of Motif loop that can manage every kind of events. Indeed Motif is the basis level of definition for the two kinds of libraries. We present on figure 8 the different levels of a window in CoDeMo and their associated graphic Object Class.

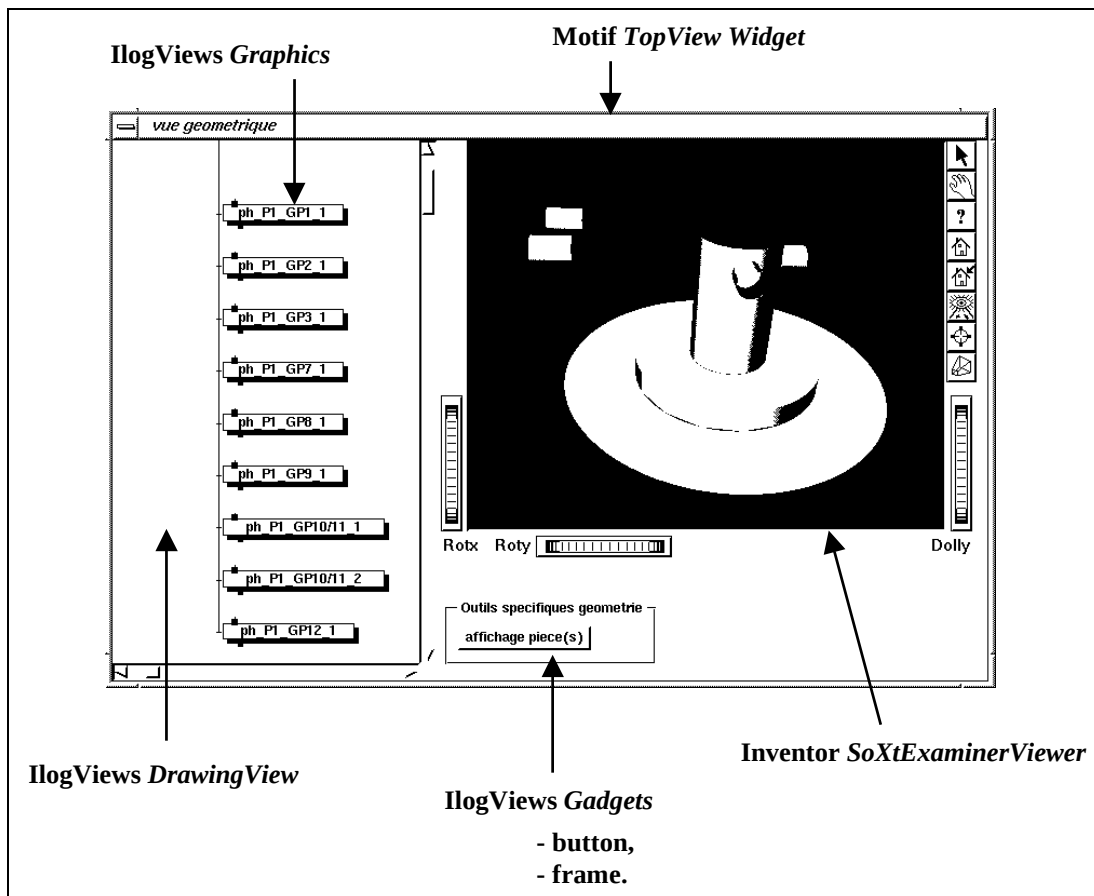


Figure 8: Mix of Motif, Inventor and IlogViews object class for the graphic interface.

1.3. Connection to specific trade applications

So far, every actor is able to watch the design state of the product and to have it evolved integrating new data. Nevertheless, this operation can only be made if an actor has new data or new constraints to provide on the product. These new data are calculated or partially defined with their own specific applications (specific trade applications), it means with their own methodology, their own tools and sometimes their own software. Every information is treated with trade knowledge and brought back to the Product Model as a new data.

To extract the existing data and get out the needed information, an actor can just edit all features associated to components with the GUI previously presented. However an automatic extraction makes easier this task. Thus, the actor has not to edit each needed datum any more, CoDeMo provides him a text file that can be read by his own software.

Figure 9 presents one connection between CoDeMo and *Copest* [2]. In this example, stamping actor describes the shaft, decomposing the Product Model in the Stamping view. He uses first a *TooledCut* feature to describe the problem. This new data has a 3D graphic representation; therefore CoDeMo delivers a text file with all information about it. In this case, the information is the nominal values of the representation. Secondly, the actor can execute *Copest* with the “Copest” button in order to read the previously extracted file. *Copest*, which is a specific stamping application, realises now the “habillage” of the tooled cut. As result, it adds the draft angles, the die split, the tooling thickness. After the calculation it returns a result file with forged cut dimensions of the shaft. In the last step, stamping actor describes the shaft with a *ForgedCut* feature associated to a new component in the Stamping view. Forged cut information is thereby read in the *Copest* result file, *ForgedCut* 3D representation is then displayed.

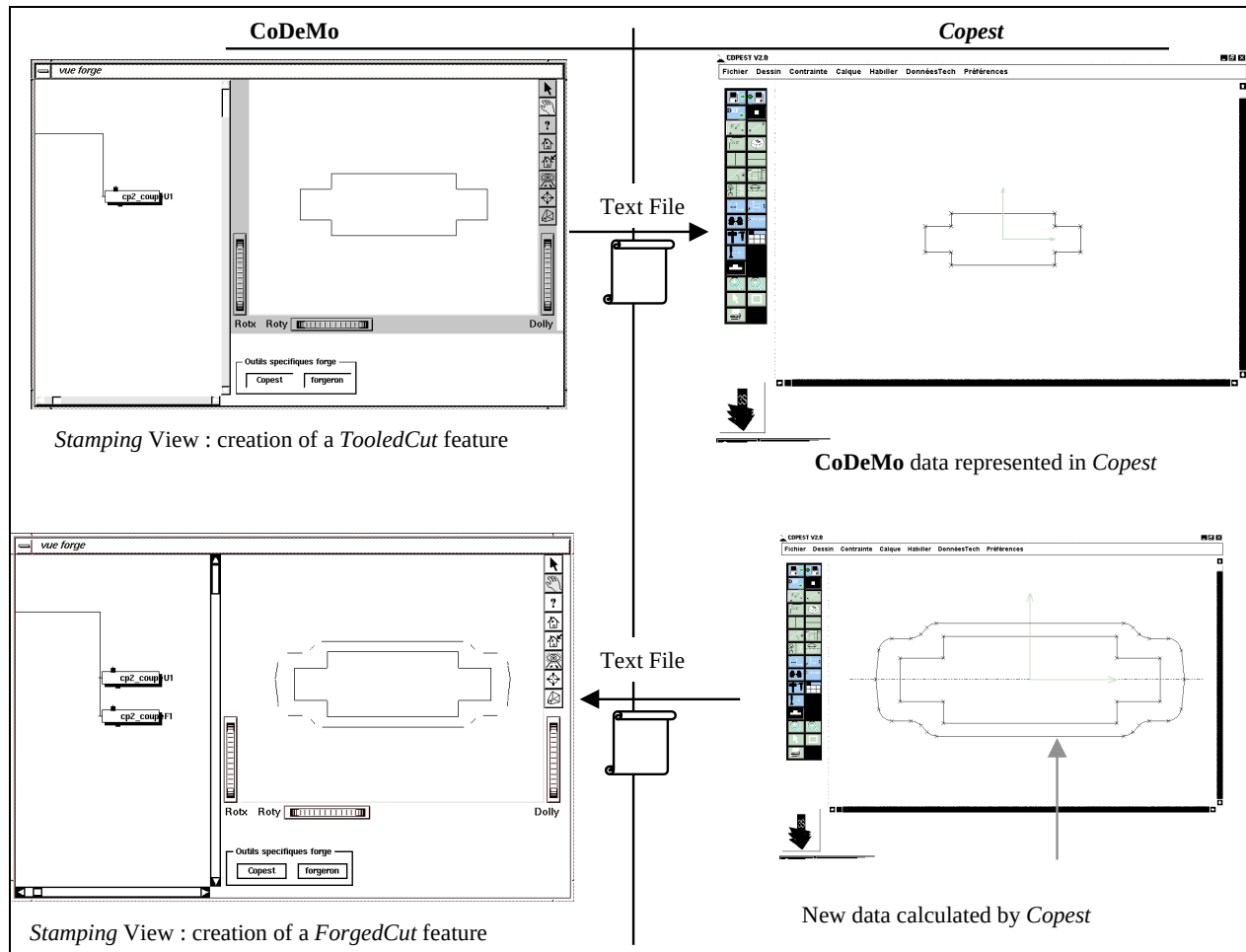


Figure 9: Connection between CoDeMo and Copest, a stamping specific application.

Several connections with specific applications have been already developed and obviously, the extraction procedure has to be customised for each ones. Moreover, such a connection is easy when file format is known. Thus, the programming of the extraction procedure consists in looking for the needed data in the shared database and writing them in a text file. When the format file is not known the programation consists in writing an interface between CoDeMo and the specific trade applications. In such a case it would be very interesting to study the creation of neutral format as STEP [20].

2. Data management

During the design process, external actors associate feature to components, links and relations in order to create the Product Model. Step by step they define and constrain the problem, however, a lot of heavy tasks cannot be realised by a human actor. They are achieved by the internal actor and are presented in three groups:

- the notifications of data through a formal exchange network,
- the translation of data during the design process,
- the propagation of data and the coherency management.

2.1. The formal exchange network: Notification functions.

The formal exchange network manages the formal co-operation between all the actors who work on the design project. As we explained in the preliminary concepts, CoDeMo has to supply the lack in co-operative design by sharing the same database among several actors. Sharing means that an actor has to be able to reach any data or any constraints of the product even if he has not created the data himself. In other words, if two client processes are running at the same time on the same server process, every kind of actions, which would be creation, modification, or deletion, in both clients has to be propagated to the server. Even further, every kind of actions has to be seen by both clients.

This formal network is partly developed using IlogBroker libraries. These libraries provide very useful object classes and a specific pre-processor. From personal object class definition, the compilation creates itself IlogBroker object classes that supply the Remote Procedure Call (RPC) between the server and all clients. For each personal header file IlogBroker pre-processor provides a *_ipl* class owning to the server and a *_itf* class owning by the client process. Nevertheless, personal header classes have to inherit of a special IlogBroker class (*Ilb_Exported* class) in order to be treated correctly during the pre-compilation (cf. Figure 10).

Using RPC/ONC communication layer, IlogBroker allows CoDeMo to be running on different platforms in the same time. For example the server can be running on Silicon Graphics whereas two client processes are running on HP and NT platforms.

Coupled to RPC connection IlogServer libraries are used to propagate all actions from server to client or from client to server. These actions are relative to the data manipulation operations that can be executed by an actor (cf. II.1.1). According to IlogServer inheritance (*Viewed Class*), some object classes can be associated to special representations. This facility is used in CoDeMo to create the different representations related to a component, a link or a relation. The notifications are in this way managed by the *ServerView* class of IlogServer. Indeed, this kind of class has pre-defined methods called for the creation (*onCreation*), the modification (*onModification*) or the deletion (*onDeletion*) of every *Viewed Object*.

As it is summarised on figure 10, CoDeMo uses *Server View* for each kind of representation (basic and 3D graphic), and three *Viewed Class* for the components, the links and the relations of the data model.

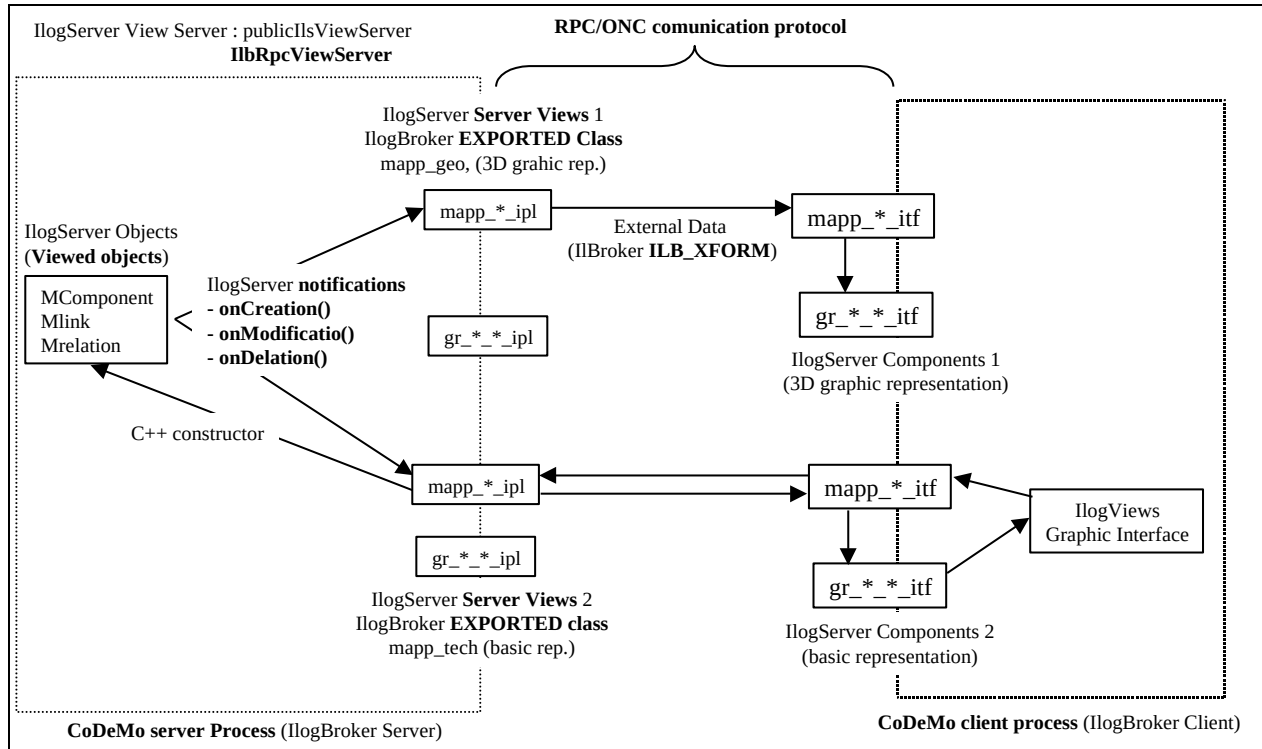


Figure 10: Uses of IlogBroker and IlogServer libraries in CoDeMo.

2.2. Feature translation

The feature translation is one of the automatic tasks managed by the internal actor. The objective is not to realise automatic design actions but only to assist external actors in achieving those heavy tasks.

As we have already presented, the Product Model is progressively built associating the knowledge model (the feature) to the data model (Components, Links and Relations). During this association, external actors are going to use features to describe the product from their own point of view, it means using their own vocabulary. In a usual communication between two foreign people who do not speak the same language words and sentences are translated in order to be understood by both participants. Like this phenomenon of comprehension, each feature used in the Product Model has to be translated in order to be understood by every design actors. For example, let *PeauArbre* be a feature that represents a functional cylindrical surface with its functional characteristics as roughness, tolerances... This feature must be translated in a *Cylinder* feature that represents the nominal dimensions of the same functional surface with its geometric characteristics as the diameter, the length... In this way the association of a *PeauArbre* feature to a component in the Ossature view involves the creation of another component with the *Cylinder* feature. The translated component is created in the right view according to the translated feature. In our example, the translated component is created in the geometric view (cf. figure 11).

Such translation process is general and thus can be used for every feature. However it obviously does not exist a translation for all features in all views of CoDeMo. That is why in a lot of cases the feature translation has to be done manually by each actor.

Figure 11 shows how a *PeauPlane* feature, which represents a plane functional surface in the Ossature view, can have several translations in the Geometric view. Indeed a plane surface can have different boundaries as a circle, a rectangle...

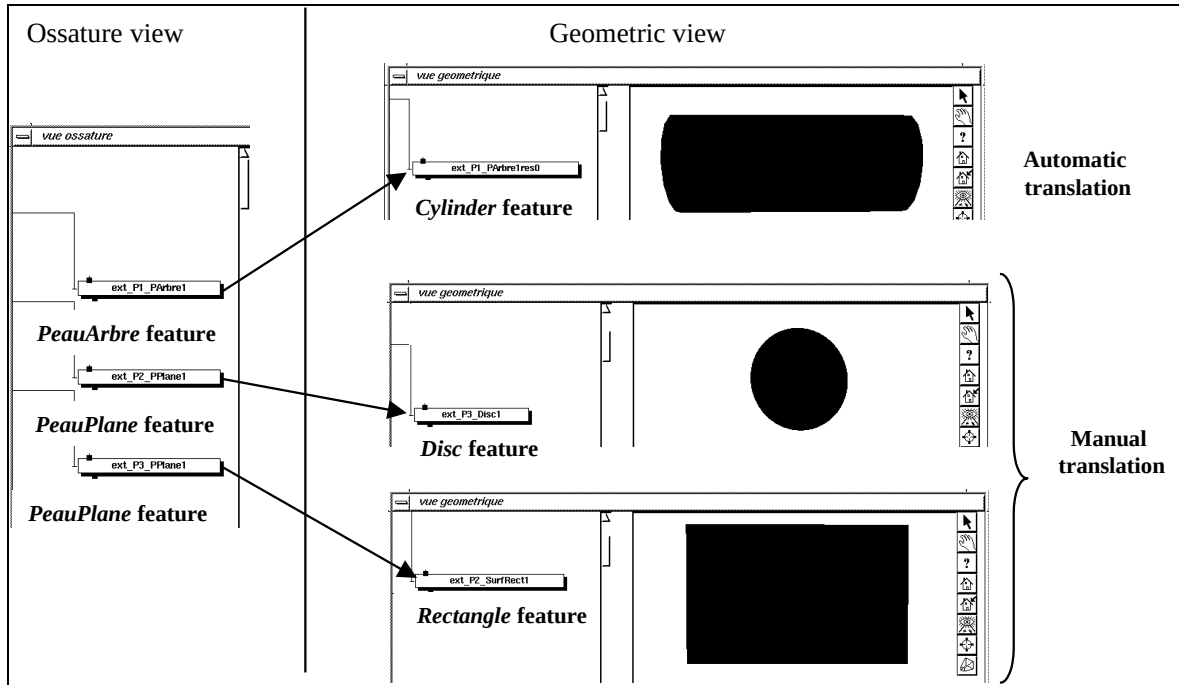


Figure 11: Feature translations as propagation of data.

2.3. Data propagation and coherency management

As the translation task, propagation and coherence management have to assist external actors. Most of current CAD systems are based on a geometric modeller and only allows the creation of relations between geometrical data (parametric geometry). In other words, they do not take into account other kinds of data and relations. One goal of CoDeMo is to manage several types of data as manufacturing, structure analysis and of course geometrical data. In such case and in a design study, there are quickly a lot of data that cannot be managed by an external actor. Indeed a human actor is not able to keep the coherency of every relation and to propagate every modification to all design data. CoDeMo is thereby developed using IlogSolver libraries. These libraries provide very useful object classes to achieve such management and propagation tasks. The use of IlogSolver libraries is done in three steps as shown on figure 12.

First data have to be defined as IlogSolver data, which are later treated such a way. Every characteristic of every descriptive feature is defined as an *IlcVarFloat* or an *IlcVarInt* variable provided by IlogSolver. These variables represent in fact an interval of respectively float or integer values. The interval can be bound: $[a..b]$, or unbound using the *IlcInfinity* variable: $[a..IlcInfinity]$. Later during the design process intervals are reduced according to constraint propagation.

Constraint features have also to be defined using IlogSolver facilities. Constraint features are defined as object class, which inherit of the *IlcConstraintI* class of IlogSolver. Thus, we have just to define the overloaded functions: *propagate()*, *post()* and *isViolated()*. These functions describe respectively the actions of the propagation, when IlogSolver has to launch the propagation and how the constraint violation is defined.

Secondly, those both descriptive and constraint features are used to construct the product model as explained in part II.1.1. In the example figure 12, we create two components associated to a *Cylinder* feature and one relation between each cylinder radius. An *Equality* feature is associated to this relation to constrain the radii. In this way, IlogSolver variables defining the two radii have to be equal; the two intervals of values have to be the same.

The third step is the automatic step managed by the internal actor and the IlogSolver manager. A predefined *IlcManager* stores all data and all constraints defined during the construction of the Product Model. Then its own function *nextSolution()* automatically propagates the constraints. The result shows that both variable intervals have been reduced according to the *Equality* constraint.

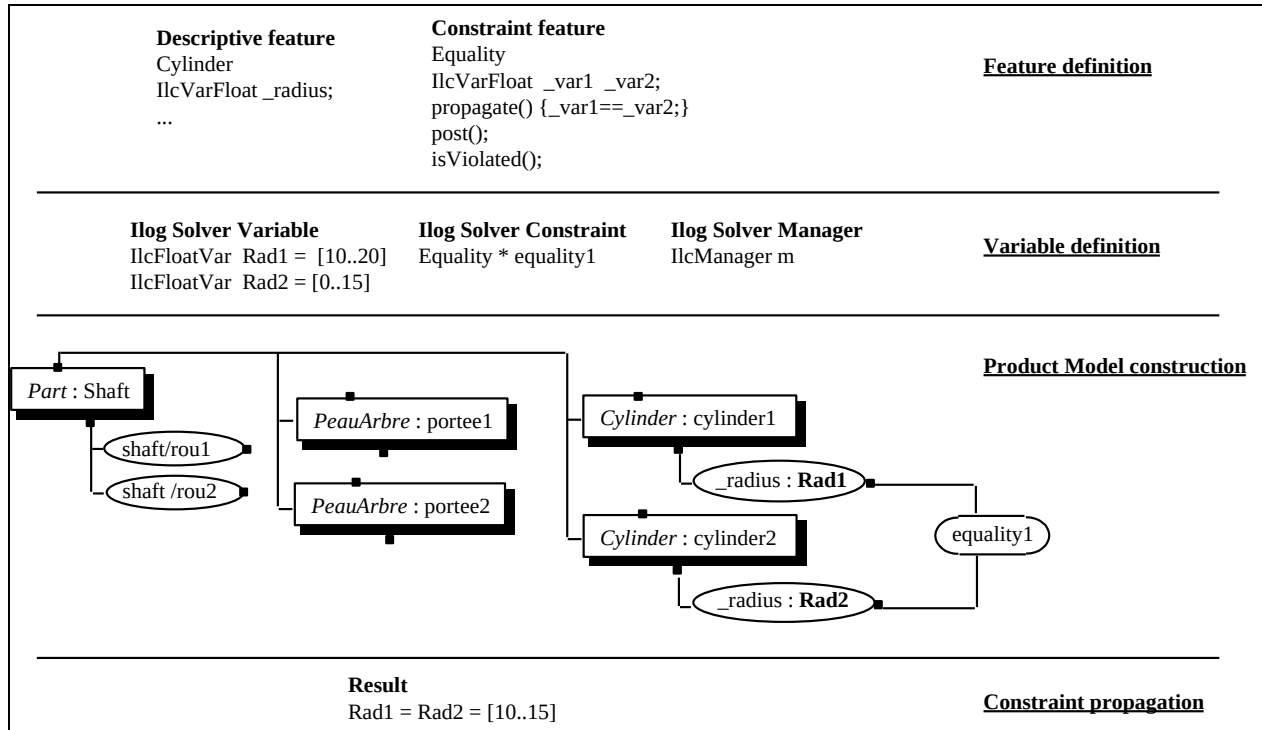


Figure 12: *Equality* feature as an example of data propagation using IlogSolver libraries.

III. PRODUCT EMERGENCE USING CoDeMo: A NEW APPROACH FOR CAD SYSTEMS

Till now, part I and II have presented the concepts and the computer developments of CoDeMo as co-operative CAD system. We have nowadays at the 3S laboratory a mock-up that can be used to do design experiments. This modeller is the support of an integrated design methodology so it is not used as current CAD systems. It is not based on geometric definition but is led by functional trade specifications. The design begins with technological choices according to technological specifications and is followed by other trade integrations in order to add new data and new constraints on the product. The geometry is only the result of such integrations and appears as the translation of functional specifications.

1. Integrated design methodology

Tichkiewitch defines the integrated design methodology as two parts in the design process [4]: initial design phase and detailed design phase (cf. Figure 13). These two phases are not really two separated phases like it is in the systematic design methodology [21] but rather mark the progressive integration of actors during the design process. Indeed, all actors are not able to bring their description and their constraints without any definition of product technology. Technological actors achieve this minimal definition in the initial phase. Other trade integrations are therefore possible and that is what is called detailed design phase.

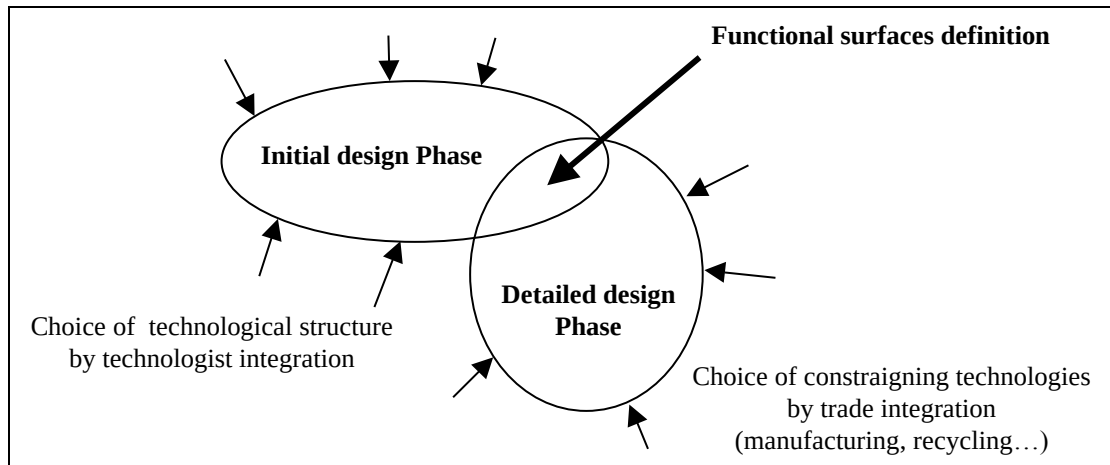


Figure 13: Two phases of integrated design methodology.

We emphasise on the non-sequentially two design phases. Indeed it would be possible that a part of the product is still in the initial phase because nobody has yet chosen a technological solution, whereas other parts were already in detailed phase and were so described by manufacturing or structure analysis actors.

2. Technological choices: the initial phase of the design

The initial phase has to bring up technological solutions according to the main functions of the product. So, technologist actors choose one solution among a list of physical admissible structures. In this way, they use specific applications based on mapping functions to structures and obtain a minimal definition of functional surfaces. These functional surfaces are necessary for the use of the chosen technological structure and are called later in the text: Usage functional surfaces.

Some software already exists on the computer market to map functions to structures. The more efficient currently is *TechOptimizer* developed by Invention Machine society [22]. Nevertheless, this product does not really deliver the minimal functional surfaces of the technology and are not neither able to be directly connected to CoDeMo. Moreover, physical principles are not used to choose solutions as in other systems [8] but just to explain the technology running.

The design example of a pin joint can be used to introduce the mapping between functions to structures. The main function of the product is then to guide a shaft allowing only one degree of freedom (only one rotation). Figure 14 shows two admissible solutions based, on the one hand on ball bearings and on the other hand on bearing. Those two solutions are the results of two different functional decompositions:

- In the first case (a), the pin joint is decomposed in the association of a ball joint and a bearing joint. The respective technologies are so, a ball bearing with two axial thrusts and an alone ball bearing.
- In the second case (b), the bearing is directly an admissible technological solution for the pin joint.

In both solutions, the shaft functional surfaces imposed by the technologies are obviously different. On one side, there are two cylindrical surfaces and two plane surfaces, on the other side, there is only one cylindrical surface and one plane surface (cf. Figure 14).

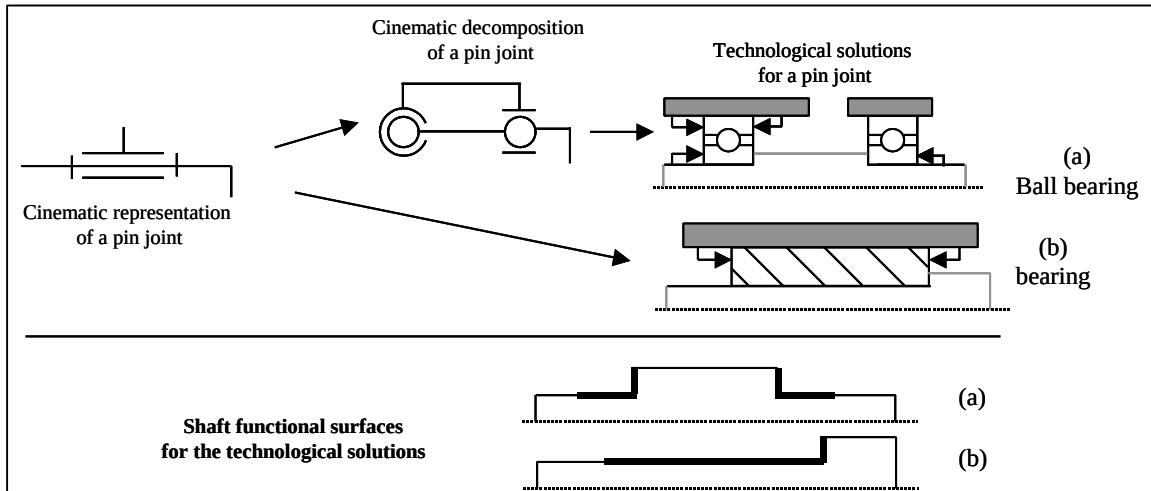


Figure 14: Two technological solutions for the pin joint, two different decompositions in CoDeMo.

Knowing the technology, technologist actors are able to construct the Product Model of the pin joint using CoDeMo. They decompose the product in their view: the Technologist view. Figure 15 presents the Technologist view of the Product Model for the ball bearing solution.

Afterwards, using the translation procedure, actors decompose the product in the Ossature view and therefore define the usage functional surfaces imposed by the technology. In this view, skin features are associated to components of the data model. The concept of skin features is defined in [23] and is used to define usage functional surfaces in the initial design phase. It is next used to define functional surfaces for trade integration in the detailed design phase.

Finally skin features in the Ossature view can be translated to define their associated nominal surfaces. It is done in order to decompose the product in the Geometric view. This is the first step of product emergence from technological specifications (cf. Figure 15).

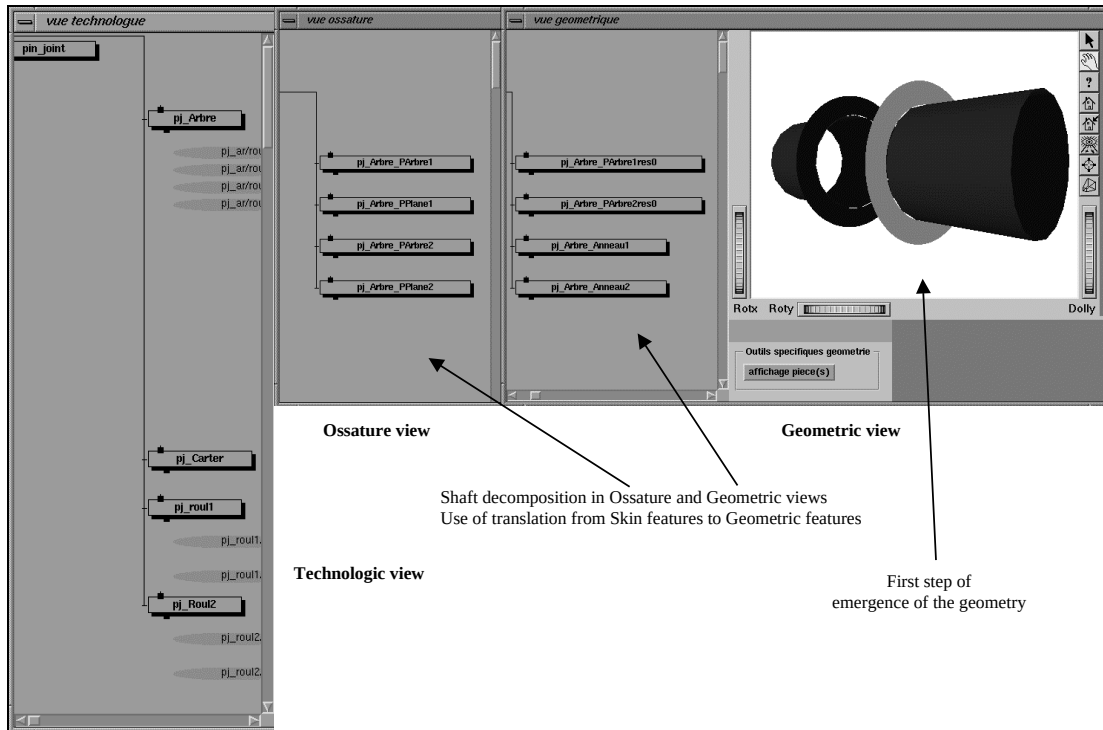


Figure 15: Product Model decomposition of a pin joint using CoDeMo.

3. Trade constraints: the detailed phase of the design

Once the choices of the technological structure and the emergence of usage functional surfaces, other trade actors are progressively going to add their constraints on the product. This is the detailed design phase.

Following the previous example we can now detail the integration of tooling actor for instance. Obviously the same integration should be done by all actors of product life cycle. We present this integration in several steps shown on Figure 16:

- (a) Tooling actor describes the product with his own point of view. Indeed, using his own vocabulary, he provides new data on the product. He has to instantiate tooling features adapted to the right description. For example, the shaft is described as *Facing*, *Sliding* and *SlidingFacing* features and their characteristics. In this step, in order to evaluate each characteristic of features, he can use specific applications like CAPP software.
- (b) Tooling actor creates the Tooling view in CoDeMo to bring his own data in the shared database. He then decomposes the shaft component of the Technologist view in several new components associated to the features chosen in the first step. According to the behaviour of each feature he also creates links and relation in the product model. For instance roughness characteristic of tooling feature, which represents the tooled roughness, has a relation with roughness characteristic of skin feature, which represents the needed roughness.
- (c) Tooling actor watches the representations of his new data displayed in CoDeMo. Tooling features are represented in the Tooling view by the basic and the 3D graphic representations. New functional surfaces appear. They are called functional trade surfaces and are represented by skin features as we did for the usage functional surfaces. In our example, tool edge imposes a radius between the cylindrical part and the plane part of the *SlidingFacing* feature.
- (d) Tooling actor decomposes the product model in the Ossature view. He adds skin features associated to components representing the previously new trade functional surfaces. In this case, a *Peau* skin feature represents the transition radius due to the cutting edge. This decomposition is based on feature translation concept and can be made automatically as far as possible.

Further, this actor should be able to define all data on the shaft process planning and even further process planning data of each part of the pin joint: casing... Once each actor has done his own description and brought his constraints in CoDeMo, the product is well defined. If it is under constrained, several solutions will be still available, it would be interesting to study which kind of new constraints is it possible to add. Otherwise, if it is over constrained, actors will have to resolve design arrangement. That could be the main goal of an actor called Project Manager.

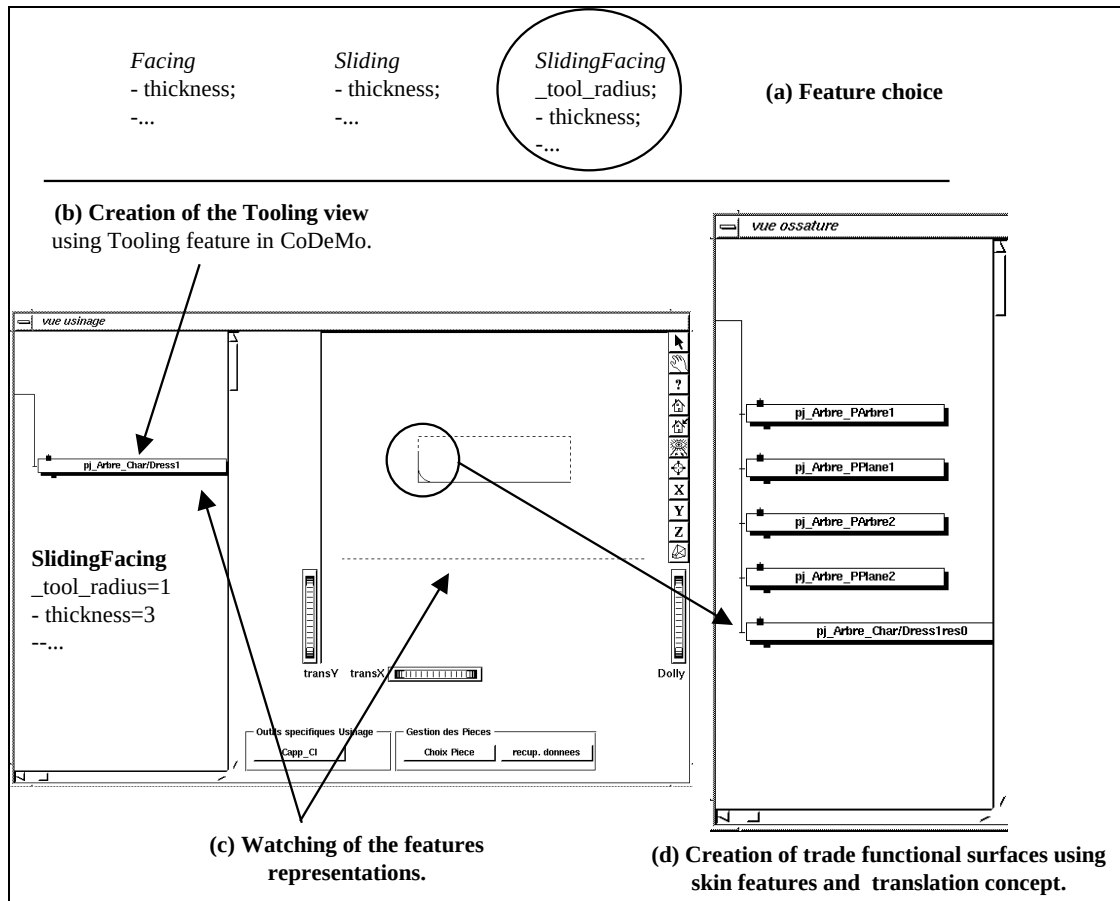


Figure 16: Integration of tooling actor in the detailed design phase: decomposition of the product model.

4. The geometry as a result : a radical change in CAD systems

Using CoDeMo as a Computer Aided Integrated Design, we emphasise on the approach by functional surfaces. Indeed each actor defines his own functional surfaces in respect to the functional specifications that would be main functions, the essence of the product, or constraint functions imposed by rules or by the design environment. The geometry is built as a result of this functional surface approach. It is the radical change compared with current CAD software even if these ones currently integrate trade modules as manufacturing or structure analysis modules. Figure 17 shows how current CAD systems use the geometry definition as a departure point whereas the same geometry is a result in CoDeMo.

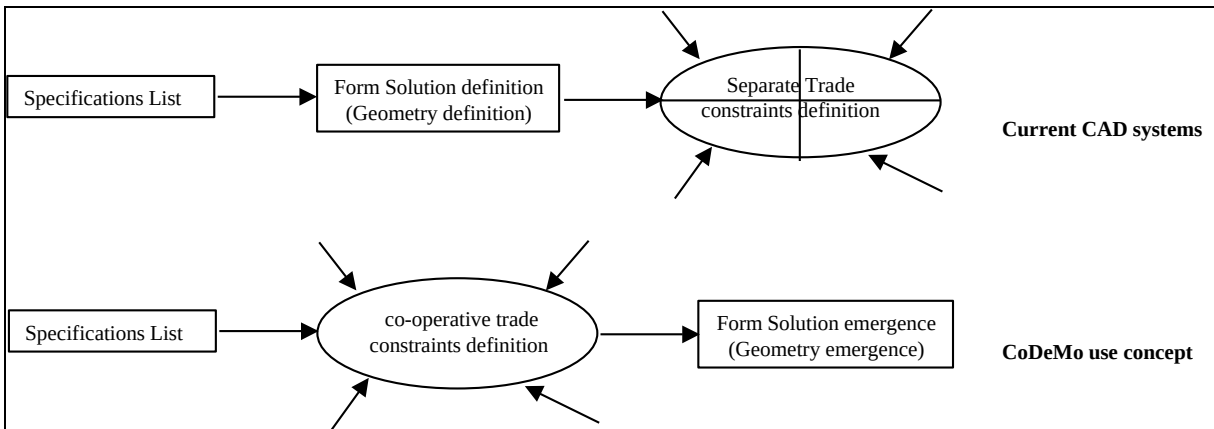


Figure 17: Radical change in the development of CAD systems.

In the *pin joint* example, we have already presented the first step of product emergence. This first step comes from the definition of usage functional surfaces brought by technological structure. The second step of the emergence comes from the definition of trade functional surfaces. In fact, new components created by tooling actor in the Ossature view are finally translated to the Geometric view. The transition radius therefore appears in the Geometric view as a *RaccordCylindrePlan* feature with geometrical characteristics (position, diameter...).

Those translations are realised automatically or not, but the origin of every geometrical data is known as a functional trade specification. Figure 18 presents the final Geometric view of the shaft in CoDeMo. We can watch the basic and the 3D graphic representation of the transition radius.

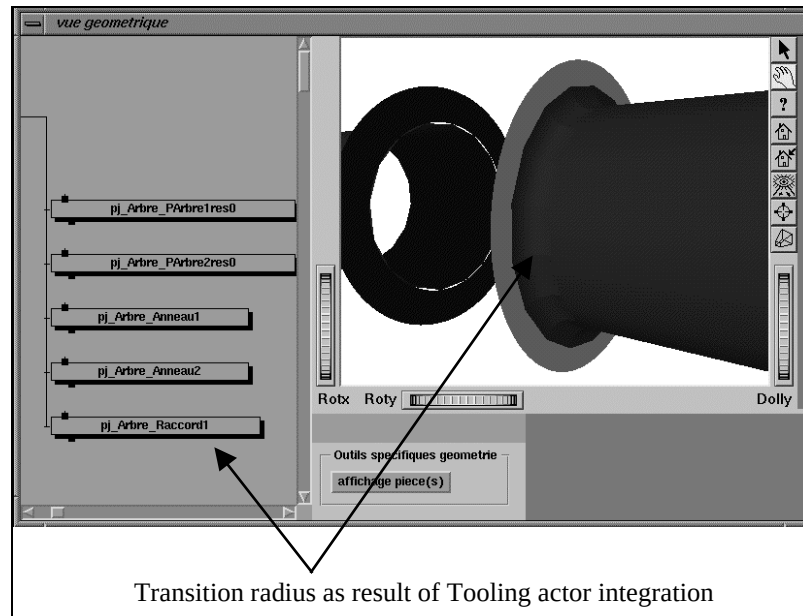


Figure 18: Final representations of CoDeMo views after the integration of technologist and tooling actor.

CONCLUSION

This paper mainly deals with the development of a new generation of CAD system used in an integrated design environment. Such system, rather than being led by the predefined geometry of the product, is based on a functional approach. Functional surfaces for the technology appear in the initial phase of the design and are completed with trade functional surfaces in the detailed design. These functional surfaces are the departure point to define the geometry of the product.

The Co-operative Design Modeller (CoDeMo) offers graphical interface to different actors who work on the design project. These interfaces are used to fill a shared database with actor's own data and own constraints. These own data and specifications can be provided by specific application directly connected to CoDeMo with neutral files. The product is therefore progressively constrained by decomposing the Product Model as the association of a multi-view data model (components, links, relations) and a knowledge model (features). This operation is available from remote computer site owing to an exchange formal network. This network connects a *server* process, which manages the shared database, and the *client* processes, which displays the Graphic User Interface.

Each graphic interface is composed of several kinds of representations (2D or 3D graphic...) and thereby each actor can understand the real meaning of each data in the Product Model. In addition, heavy design tasks as data propagation or constraint coherence are managed by CoDeMo and become therefore transparent for the software users.

CoDeMo has been developed using Ilog libraries that supply with a lot of problems as the graphic interface definition, the multi-processes communication, the data management, and the constraint propagation. These libraries are coupled with Motif and Inventor libraries from Silicon Graphics in order to implement the 3D graphic representations of data.

REFERENCES

1. Tsang J.P., Brissaud D., "A feature-based approach to process planning", Computer in Engineering Conference and Exhibition, ASME-CIE, Anaheim, August 1989.
2. Boujut J.F., Tichkiewitch S., "COPEST: A Co-ordination Tool In The Design Process Of Stamped Parts", AMPT '95, Advances in Materials And Processing Technologies pp 857, Dublin, August 1995
3. Escande E., "Modélisation objet du processus de conception dans le domaine du génie électrique : application au cas de la machine asynchrone", thèse de l'Institut National Polytechnique de Grenoble, 1996.
4. Tichkiewitch S., "De la CFAO à la conception intégrée", International Journal of CAD/CAM and Computers Graphics, vol. 9, n 5, pp 609-621, 1994.
5. Ishii k., "Modelling of concurrent engineering design", Concurrent Engineering Automation, Tools and techniques, edited by Adrew Kusiak, ISBN 0-471-55492-8, Wiley inter-science publication. 1993.
6. Gabbert U., Wehner P., "The Product Data Model as a pool for CAD-FEA data", Engineering with computer, vol. 14, pp 115-122, 1998.
7. Schulte M., Weber C., Stark R., "Functional features for design in mechanical engineering", Computer in Industry, n. 23, pp 15-24, 1993.
8. Tichkiewitch S., Roucoules L., "Innovative design methodology", Integration of process Knowledge into Design Support, ISBN 0-7923-5655-1, Kluwer Academic Publishers, 1999.
9. Troussier N., Pourroy F., Tollenaere M., Trebucq B., "Mechanical models management in engineering design", Integrated design and Manufacturing in mechanical engineering '98 Processings of the 2nd IDMME Conference held in compiègne, france, 27-29 May 1998 Kluwer Academic publisher.
10. Lange M., Kjelberg T., "On motivation of Feature Selection in Feature based Modeling Systems", Integration of process Knowledge into Design Support, ISBN 0-7923-5655-1, Kluwer Academic Publishers, 1999.
11. Tichkiewitch S., "Specification on integrated design methodology using a multi-view product model", ESDA Proceeding of the 1996 System Design and Analysis Conference, PD-Vol 80, 1996.
12. Débarbouillé G., "Design data structure - version 1.0", Rapport interne, Matra Datavision, 1994.
13. Brissaud D., Paris H., "A model for evaluating the quality of the fixturing of the part from a machining process planning perspective", Int. Jour. of Advanced Manufacturing Technology, Vol. 14, Num. 3, pp 172-179, 1998.
14. Sohlenius G., "Concurrent Engineering", Annals of the CIRP, vol. 41, n°2, pp 645-655, 1992.
15. Aldanondo M., Rougié S., Veron M., "Expert configurator for concurrent engineering Caméléon software and model", Journal of Intelligent Manufacturing, vol. 11, pp 127-134, 2000.
16. Lutters, D.; Streppel, A.H.; Kals, H.J.J.; "Product information structure as the basis for the control of design and engineering processes"; In: Proceedings of the 29th CIRP International Conference on Manufacturing Systems, pp. 167-166; (1997), Osaka.
17. Huang G.Q., Mak K.L., "Web-based morphological charts for concept design in collaborative product development", Journal of Intelligent Manufacturing, vol. 10, pp 267-278, 1999.
18. ILOG, Bâtiment ORSUD, 3-5 avenue Galiéni 94 257 Gentilly Cedex, France. <http://www.ilog.com>.
19. Shah J.J., "Assesment of Feature Technology", Computer Aided Design, vol 23, n°5, june 1991.
20. International Standart Organization, STandard for the Exchange of Product data (STEP), ISO Standart 10303.
21. Beitz W., Küttner K.H., "Handbook of Mechanical Engineering", springler verlag, 1994.
22. Invention Machine France, 72, quai des Carrières 94220 Chateron-le-pont. <http://www.inventionmachine.com>.
23. Tollenaere M., Belloy Ph., Tichkiewitch S., "A part description model for the preliminary design", Advanced CAD/CAM Systems - State-of-the-art and future trends in feature technology, pp 129-143, Chapman & Hall , Ed. Soenen, 1995