Distributed data model for collaborative CAD environments based on design history

Carlos A.M. Barbosa[1], **Bruno Feijó**[1], **Marcelo Dreux**[2], **João Bento**[3], **Rubens Melo**[4], **Sergio Scheer**[5]

[1] Intelligent CAD Laboratory, Department of Computing, PUC-Rio, [barbosa,bruno]@inf.puc-rio.br
[2] Department of Mechanical Engineering, PUC-Rio, dreux@mec.puc-rio.br
[3] DECivil, Instituto Superior Técnico, Technical University of Lisbon, joao@civil.ist.utl.pt
[4] Group of Technology in Database, Department of Computing, PUC-Rio, rubens@inf.puc-rio.br
[5] CESEC/TC/UFPR, scheer@cce.ufpr.br
PUC-RioInf.MCC22/99 November, 1999

Abstract: This paper presents a distributed dynamic data model that is aligned with the concepts of design as problem solving and design history. It is proposed a model for distributed CAD data that allows teams to work cooperatively, being able to access and change information in distributed engineering database systems, at run time. The virtual prototype is distributed over different networks with different platforms, operating systems, design teams and non-CAD users. The artifact specification is defined in a dynamic process, and a tree is used to provide user interaction to establish the object model for the sought artifact. The description regarding each object is stored as tuples in tables of conventional and heterogeneous databases. These objects are organized in such a way to allow the description of the design evolutionary aspects. Besides, the artifact properties can be associated with certain subject of interest or operational flow within the organization. Several associated activities within the design productive process (organization hierarchy, planning, process workflow, artifact specification, etc.) are integrated in the architecture of the proposed CAD environment.

Keywords: distributed database, collaborative work, design history, design environment, product modeling

Resumo: Este artigo apresenta um modelo dinâmico de dados distribuídos que está alinhado com os conceitos de *design* como solução de problemas e história de *design*. É proposto um modelo para dados distribuídos de *CAD* que permite equipes trabalharem cooperativamente, sendo possível acessar e alterar informações em um sistema de banco de dados distribuído, durante o processamento. O protótipo virtual é distribuído em redes com diferentes plataformas, sistemas operacionais, equipes de *design* e não usuários de *CAD*. A especificação do artefato é definida em um processo dinâmico e uma árvore é usada como interação, para o usuário estabelecer o modelo de objetos para o artefato procurado. A descrição referente a cada objeto é armazenada como tuplas em tabelas de banco de dados distribuídos e heterogêneos. Estes objetos estão organizados de maneira a permitir a descrição dos aspectos evolutivos do processo de *design*. Além disto, as propriedades do artefato podem ser associadas a um determinado tema de interesse ou fluxo operacional da organização. Várias das atividades associadas ao processo produtivo em *design* (hieraquia da organização, planejamento, fluxo de processos, especificação do artefato, etc.) estão integradas na arquitetura proposta para um ambiente de *CAD*.

Palavras chave: banco de dados distribuído, trabalho colaborativo, história de *design*, ambiente de *design*, modelagem de produto

Introduction

The aim of virtual prototyping is to build a full virtual artifact in such a way that design and manufacturing problems are anticipated and discussed within a cooperative and distributed work environment, which embraces all the departments of a company. Virtual prototyping is much more than building a complete 3D model of an artifact. A virtual prototype works as a spatial and temporal database that can be built and queried by anyone in the enterprise on a heterogeneous computer network. The objects of a virtual prototype have several types of attributes, such as geometric values, design intents, manufacturing specifications, cost data, part numbers and references to documentation. Furthermore, these objects should be defined in the context of workflow management (Georgakopoulos et al., 1995), organizational engineering (Kerzner, 1997) and requirements traceability (Gotel and Finkelstein, 1994) (Ramesh et al., 1995).

Figure 1 illustrates a distributed CAD environment, where the virtual prototype is distributed over different networks with different platforms, operating systems, design teams and non-CAD users. The design process requires collaborative work amongst several teams. After the dynamic stage of the design process is completed and as soon as manufacturing drawings are approved, components can be stored in a central database containing the Product Structure file and the STEP file. Essentially, the Product Structure is a collection of pointers to 3D models and 2D drawings. A STEP file is more than a format for data exchange, it can be browsed (as a text or as a 3D object) by anyone on the intranet. However, during the design process, data are manipulated over different database systems and networks in a highly dynamic way that goes beyond simple web browsing. A model for distributed CAD data should allow teams to work cooperatively, accessing and changing information in distributed engineering database systems at run time.

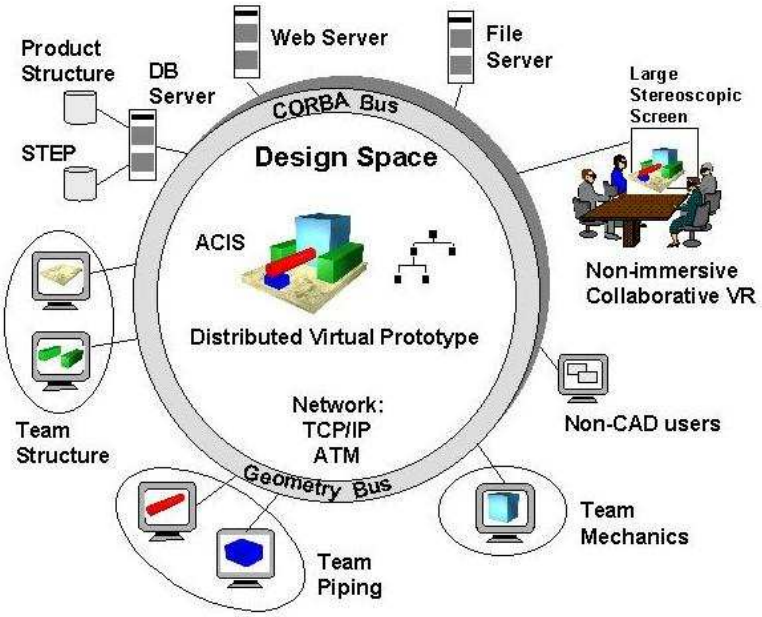


Figure 1 - Distributed CAD environment

The present work proposes a data model for virtual prototyping based on geometry bus and object-distributed computing with the focus on the design history. This paper is organized as follows. Firstly, the object model and the concept of design history are presented. Secondly, a distributed system is proposed as a component architecture, which can be easily customized to work with different middleware for distributed objects (e.g. CORBA or COM) and different programming languages at the interface level. This distributed system considers data stored in different database systems, which are distributed on different networks. Thirdly, an integration model for CAD environments is proposed based on the paradigms of workflow management and organizational engineering. Fourthly, the concept of geometry bus is presented as the cornerstone for interoperability and the basic foundation for collaborative work with geometric data. Finally, conclusions and recommendations for further work are presented.

The Object Model

In the proposed model, the link between two objects is also an object as defined by the following class:

```
class Link
  source_tag     //tag of the source object
  target_tag     //tag of the target object
  relation       //IS_A, PART_OF
  context        //ELECTRICAL and/or MECHANICAL and/or …
  start_time     //the time when the link is created
  end_time       //the current time or the time when the link is
                 //removed
```

Figure 2 illustrates two types of link (*is-a* and *part-of*) and the visualization of objects within a specific active context. In this paper only *is-a* and *part-of* relations within a hierarchical tree are used. However, links and objects can form cyclic graphs with generic constraint relations as proposed by Feijó et al. (1998) In the proposed object model, the following assumptions are made:

(1) The link is valid only in the interval `[start_time, end_time)`;

(2) Objects can only be visible if their links correspond to the current active context (electrical, mechanical, mechanical and electrical, …);

(3) Attributes and methods (defined by the users) are stored in textual form in database tables, which can be distributed amongst different database systems and networks;

(4) The position of an object in the tree is recovered into the working memory when the object is retrieved from the database tables;

(5) An object can be part of more than one hierarchical tree (*e.g.* the same object can be used in the specification of two or more artifacts).

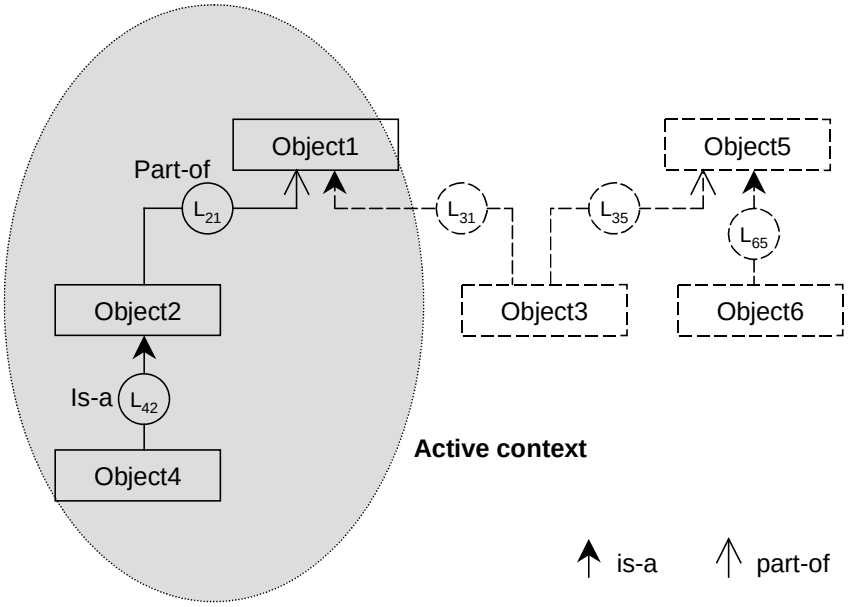


Figure 2 - Links and active context

In the proposed model, objects are associated to attributes, which are classified according to the properties they represent. Furthermore, the attributes are also objects. In the CAD environment, the set of objects that associates attributes to an object in the root of a hierarchical tree is called a **design entity** (or design agent, or simply, entity). *Figure 3* illustrates the proposed model for design entities, which have two main types of attributes: **general properties** and **solids**. In this figure, the double-arrow head represents a *has-a* relation with multiple objects. The relation *has-a* is the inverse of the relation *part-of*. In this model, general properties can be essential or optional (*PropOptional*), and solids can be geometrical, topological or optional (*SolOptional*). Everything in the CAD environment is a

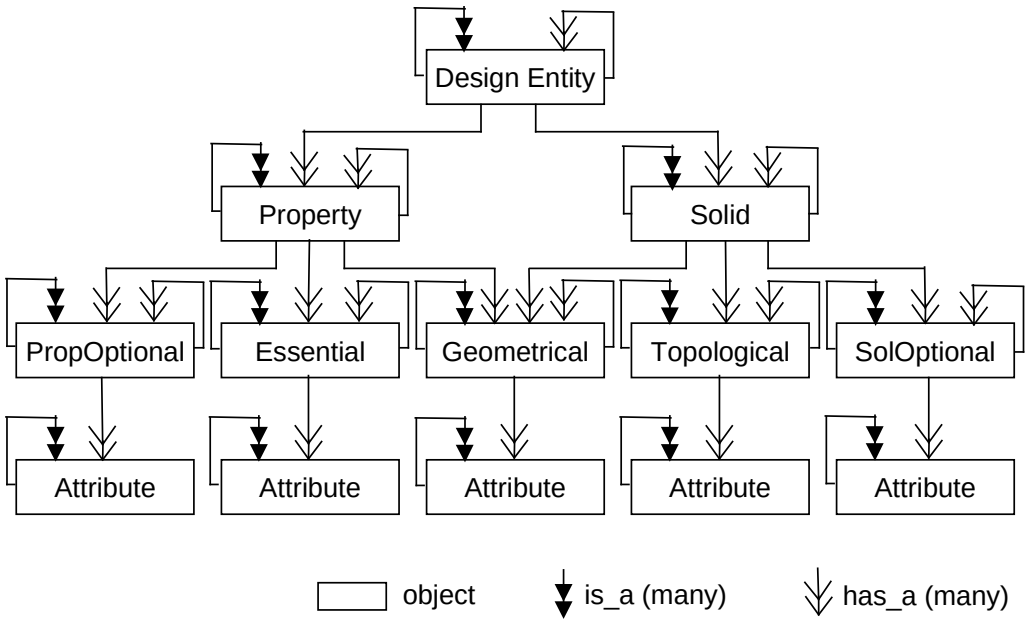


Figure 3 The model of the design entities

design entity, from the designer up to a 3D component displayed on the screen. A designer, for instance, can be represented in the environment by an entity with a few properties *Essential* and no 3D representation (*i.e.* there are no attributes *Solid*). Although this model creates a very large number of objects, it is quite adequate to trace the evolution of attributes and relationships, a necessary condition for future developments towards requirements traceability. Also this data model is adequate to deal with the problem of function and form presented in the next section.

Design History

Design was first identified with problem solving by Herbert Simon (1969). According to his approach, a state space represents all possible states of the problem (i.e. all possible problem descriptions) that need to be considered when a solution is attempted. Besides, he claims that in practice it is possible to compute all the state space.

Design as problem solving is a search process within a state space called the design problem space (DPS). In the context of traditional search, design knowledge is to be expressed in terms of goals and operators. Such a plain concept of searching does not directly address the characteristics of the design problem (Maher, 1990). The difficulties in this case are related to the variations of goals during the problem solving process and the problem of predetermining the relevant operators. Therefore, the notion of design as problem solving needs to be presented in a broader sense. For instance, the proposal of *design as exploration* (Simithers and Troxell, 1990) can be understood as a meta-search process within the design problem space. It restores the concept of meta-planning for design perceived by Herbert Simon (1969) more than three decades ago.

Generally speaking, the implementation mechanisms for design as problem solving do not need to use the classical binomial goal-operator or even traditional planning techniques. However, whatever the case, the minimum requirement is the conformance to the general principles of improving upon blind trial-and-error search (Minsky, 1988), that is:

(1) the progress principle (i.e. the ability to detect when progress is being made);

(2) goals and subgoals (i.e. the decomposition procedure that reduces the problem space);

(3) the use of knowledge (*i.e.* if one knows how to solve a problem, then one can entirely avoid the search).

The object model presented in this paper is flexible enough to accommodate generic implementations of design as problem solving. Furthermore, the object-oriented approach adopted by the present work facilitates the solution of well-known design problems.

From the design point of view, goals can be decomposed in terms of **form** (mainly physical attributes) or **function** (functional specifications describing the functions to be performed by the form). The question form *vs* function may pose a dilemma for goal decomposition. However, in a problem solving approach to design this situation does not occur because representations of goals may capture both the notions of form and function

(Maher, 1990). The design entities presented in *Figure 3* can be easily used by orthogonal views in order to allow reasoning on form and function, as shown in *Figure 4*. In fact, the plane of the object tree is the basic view amongst several other possible views. The active contexts mentioned in the previous section are particular design views within the basic view.

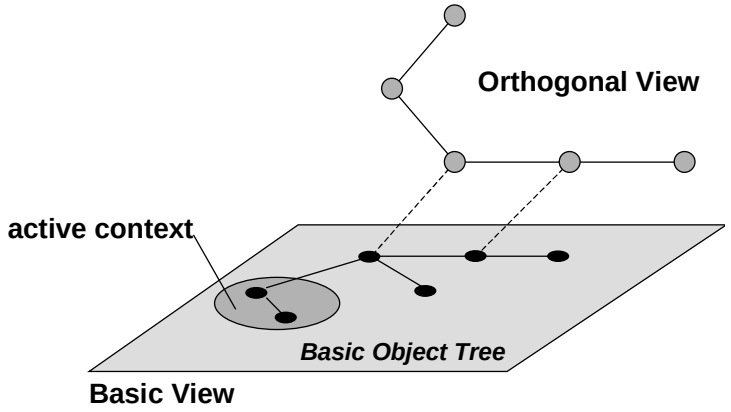


Figure 4 - Design views

The artifact is a collection of design entities with a well-defined design history containing the evolution of its entire specification. Each design entity has a **design history** defined by the following triple:

$$History_i(t) = \langle \{State\}_i, \{Version\}_i, Entity_i \rangle, \qquad t = current_time$$

that is, the design history of an entity has a set of states and a set of versions. In each instant of time, a design entity is associated with a unique version and a set of states representing its life cycle. A designer can go back in history and create new versions of a component, depending on his/her degree of privilege. However, some entities in *History* may be locked in order to guarantee consistency. In the early stages of the design history, there is only a set of general specifications.

The following definitions are important for the proposed data model:

Def.: The **design problem space** (DPS) is the set of all possible design histories.

Def.: The **artifact state** $T(t)$ is given by the set of all histories at time t, that is:

$$T(t) = \bigcup_{i=1}^{n} History_i(t)\,, \text{ where n = total number of entities}$$

Def.: **Input specifications** (*Spec*) are the initial artifact state, that is:

$$Spec = T(0)$$

Def.: The **artifact description** (*Desc*) is the artifact state at time t_{end} when all requirements are satisfied:

$$Desc = T(t_{end})$$

In this context, design is an evolutionary process that starts with the input specifications *Spec*, generates a kernel idea in the early stages of the process and refines it (by decomposition, generation or transformation) towards the artifact description *Desc*.

Attached Predicates

Design entities are mutable objects in the sense that their properties may change with time passing. The ability to support mutable objects implies the automatic propagation of changes occurring in the mutant object to those other objects that have references to it. This propagation of changes is implemented by procedures attached to attributes called **attached predicates** (Bento, 1992). These predicates can be functions defined in a procedural language (such as Lisp, Lua and C) or logic predicates defined in a logic programming language (like Prolog). Attached predicates are used for the purpose of triggering procedures on variable access, drawing on the style of *active values* and *access-oriented* programming techniques (Inference Corporation, 1985). This idea is also motivated by the new concept of active database systems (Widom and Ceri, 1995).

In this work, one of the attached predicates is called *if-changed*. This kind of predicate is a procedure that must be executed in the case of an attribute value being tentatively changed, before the assertion of the attribute value takes place. These predicates behave like "watchdogs" of the attributes to which they are attached. Another attached predicate is called *if-needed*. Whenever no default value is available locally or through inheritance, the predicate *if-needed* provides a means for computing it (Bento, 1992).

The Distributed System

Figure 5 presents the context diagram of the proposed distributed system where the whole tree of objects is a virtual structure corresponding to the union of database tables distributed on the computer network. When a subset of this tree is requested by a client, only a simple list of links and object lables are stored in the main memory, with no need for complex structures and large amounts of data. In this case, each node of the tree in the main memory corresponds to a unique row of a particular conventional database table containing all the data of the object. In this distributed system, the user can always point the mouse to a 3D virtual component built by someone from another design team and ask for the value of a particular attribute. In this case, the client looks for a server capable of retrieving this value from one of its database tables. Another example is when the user performs a particular geometric operation on a component that belongs to another design team. In this latter case, if the transaction is allowed, the client gets the geometric method from a remote database table with the help of the server and executes it locally.

The distributed system uses only the basic architecture of CORBA, because the communication layer could be easily substituted by an alternative to CORBA, such as COM. Therefore, CORBA's naming services are not used.

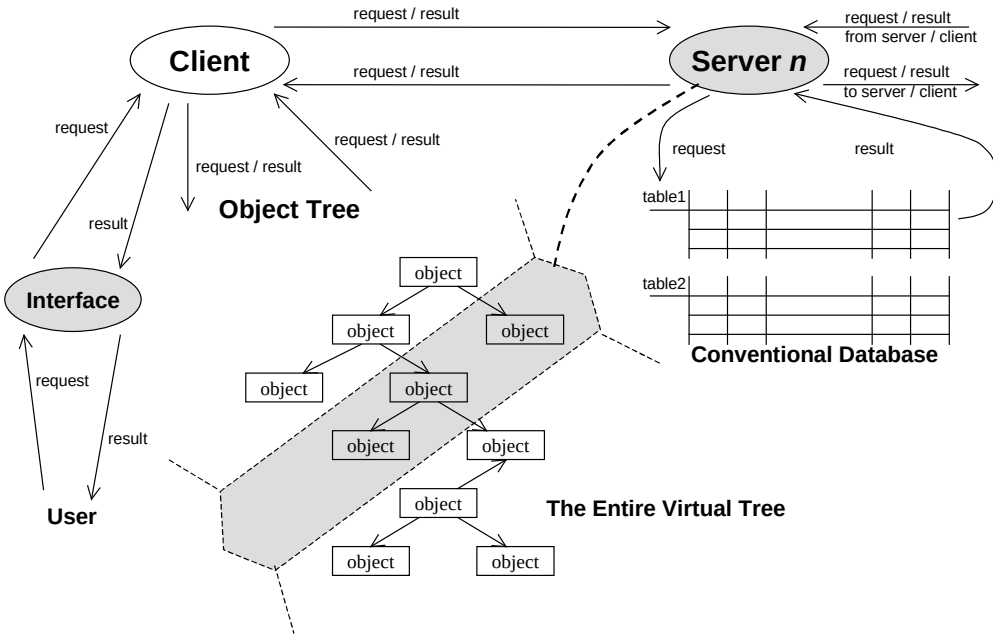


Figure 5 Distributed system diagram

In the proposed system, both IDL interfaces for the server and the client (`SOrb` and `COrb`) have their methods with the same signature, that is:

(1)
```
interface SOrb
{ string ReceiverMsg(in string input_string); };
```

(2)
```
interface COrb
{ string ReceiverMsg(in string input_string); };
```

The following piece of code in standard CORBA file illustrates how the reference to the object `SOrb` (located in the server) is obtained from the client.

```
CORBA_Object_var obj = orb->string_to_object(SOrb_IOR);
SOrb_var sorb = SOrb::_narrow(obj);
```

where `SOrb_IOR` corresponds to the IOR (Interoperable Object References) for the server in question. The IOR is stored in a file that is a common reference to all objects on the network.

Very few lines of code depend on CORBA and the mechanism of accessing the methods of remote objects is quite simple. In the client, the pointer `sorb` to the remote object `SOrb` located in the server gets the method **`ReceiverMsg`** of this object through an input string, that is:

```
char* result_string = sorb->ReceiverMsg(input_string);
```

In a similar way, the object `COrb` located in the client is called from the server with the help of the pointer *corb* in the following piece of code:

```
char* result_string = corb->ReceiverMsg(input_string);
```

The utilization of a string (*`input_string`*) is simpler and more general than the use of several CORBA structures of type Any. Strings also facilitate additions of new structures at

run time. This string is the base for the communication amongst the objects and is called **communication string** in the present work.

In a client or a server, the method **ReceiverMsg** passes its parameter *input_string* to the method **ProcessMsg** that processes the message contained in the communication string. In **ProcessMsg**, a case structure directs the message *input_string* to one of the design entities defined in *Figure 3*. All design entities have a similar way of processing the message. The following piece of code reveals this mechanism:

```
ProcessMsg (PMsg msg)   //PMsg is a pointer to the class Msg
                        //that is instanced for the current
                        //communication string
  obj = msg->GetOOpr() // GetOOpr() gets the type of operation
  switch(obj)
  { case HISTORY:     History->ProcessMsg(input_string);
                      break;
    case ENTITY:      Entity->ProcessMsg(input_string);
                      break;
    case ATTRIBUTE:   Attribute->ProcessMsg(input_string);
                      break;
  .....  }
...
```

The use of the case structure above mentioned is the key for supporting dynamic distributed objects in CAD environments. In the proposed distributed system, the user manipulates objects, attributes, and methods by creating a string that contains all information needed to store data in conventional database tables to retrieve them from those remote tables. Therefore, the distributed tree of objects can be created or modified at run time with no need for complex typed languages like TclDii (Almasi et al., 1999) or CORBA dependent solutions (Houlding, 1998). *Figure 6* illustrates the proposed organization of the database tables for the three main objects HISTORY, ENTITY and ATTRIBUTE.

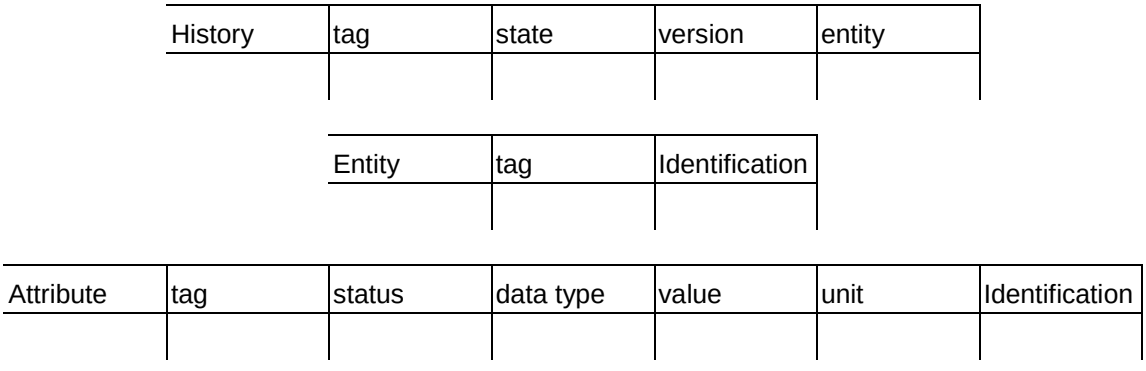


Figure 6 Database tables for the objects HISTORY, ENTITY and ATTRIBUTE

The tag of the database tables has the following format:

host_domain_object_entry

where: *host*: host computer where the object is stored.

domain: classification that depends on how information is organized in a specific project, such as knowledge area, working group, … .

object: type of object, such as entity, attribute, method, … .

entry: sequential number of the specific object within the table

The communication string is organized in fields that are identified by tokens. These tokens can be short single labels or compound labels of the form #ifld#N#M. The following examples illustrate the format and types of use of the communication string:

(1) Insert an object ENTITY called Piston Rod with a generic functional description:

```
#mopr insert object #ohst davinci #odmn icad #oobj entity
#ifld#0#0 Piston Rod #ifld#0#1 piston to generate force for
the vehicle
```

(2) Insert the attribute mass as a component of an object ESSENTIAL, with a value of 100.0 kg and within the hydraulics and mechanics contexts:

```
#mopr insert component #otag davinci_icad_essential_9
#ifld#0#0 float #ifld#0#1 100.0 #ifld#0#2 kg #ifld#1#0 mass
#ifld#1#1 mass of piston #ifld#2#0 HYDRAULICS, MECHANICS
```

(3) Get an object ENTITY:

```
#mopr get object #otag davinci_icad_entity_5
```

The class **`Msg`** manages the operations of inserting and retrieving fields in the communication string.The class **`Msg`** organizes the message fields in lists and the string is only retrieved when the message is sent to another host. The communication string is generated with the help of a text module that uses an interpreted language called Lua

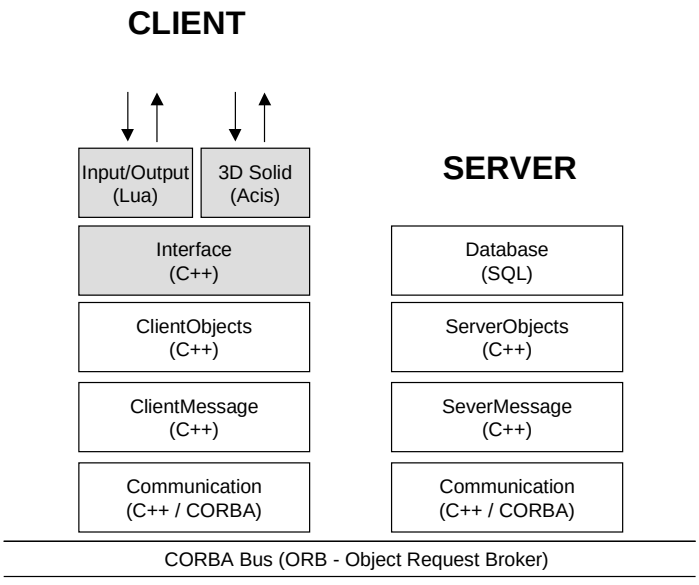


Figure 7 - Component architecture of the distributed system

(Ierusalimschy, 1996). Some processes written in Lua establish the communication between the environment interpreted by Lua and the compiled environment in C++.

Figure 7 presents the component architecture of the distributed system, where any component can be substituted. For instance, CORBA could be substituted by COM, any other database system could replace the SQL component, and the language Lua for the text module could be replaced by Lisp.

An Integrated Model for CAD Environments

Figure 8 presents an integration model for CAD environments, where double-arrow heads indicate a list of objects (i.e. a set of design entities). All relationships in this model are *has-a*, except *Specification* that is a *Product* (*i.e.* it has an *is-a* relation). When a specification has geometric and topological attributes, it has a solid that can be visualized through the 3D solid module.

Every team has a group of designers and is associated to one or more plans of activity.

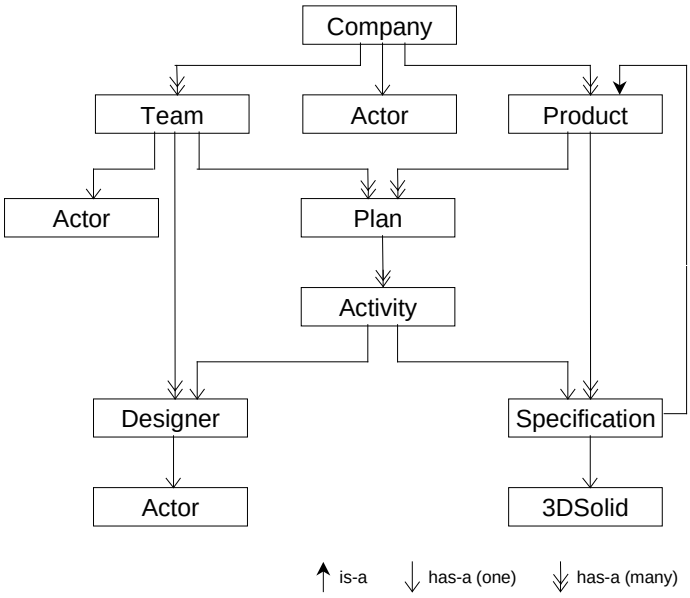
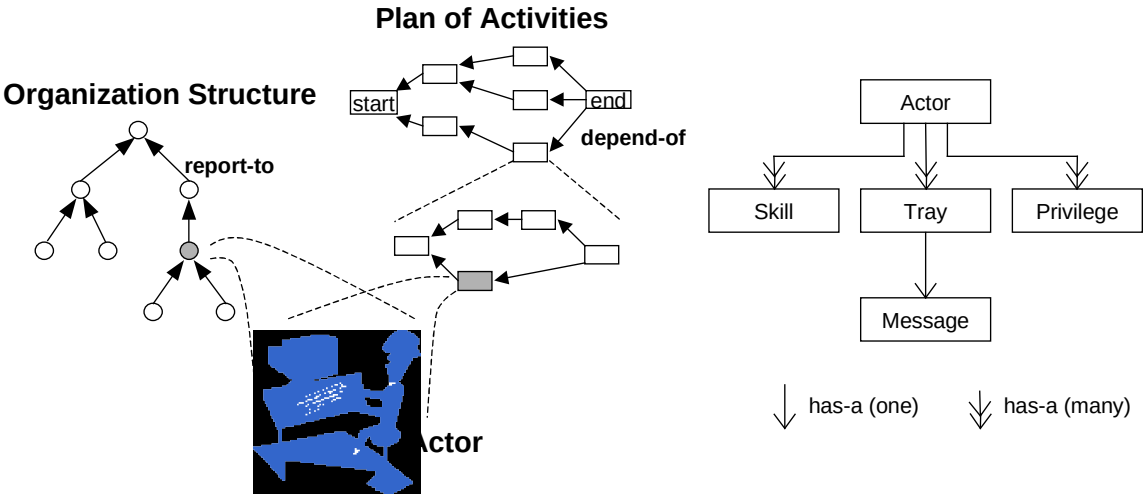


Figure 8 - Integration model

Moreover, actors assigned to *Team*, *Designer* and *Company* define a hierarchical structure called the **Organization Structure**. *Figure 9* shows that any particular *Actor* can be viewed as part of the Organization Structure (*i.e.* he/she reports to someone else) and is part of the resources allocated to a certain activity. *Figure 9* also presents the model of an actor. This view of the integration model is based on the workflow paradigm and the concepts of organizational engineering. The Virtual Design Team draws the idea behind this twofold view of an actor from the work (Levitt et al., 1994).

At the start of the design process, each actor is assigned to a position in the team organization and to one or more activities. Actors process information items in their in-tray,

send messages to others through their out-tray and cause changes in the design history. Plans of activities are created in the very beginning of the design process and are the backbone of the entire process. *Figure 9* also illustrates the fact that any complex activity can be opened into a simpler plan.



(a) Actors, organization and plans (b) The model of an actor

Figure 9 - Actors

Geometry Bus

The present work uses ACIS (Spatial Technology, 1998) as a geometry bus, although other kernels could be used. A geometry bus consists of the following items:

(1) a standard object-oriented geometric library;

(2) a component architecture;

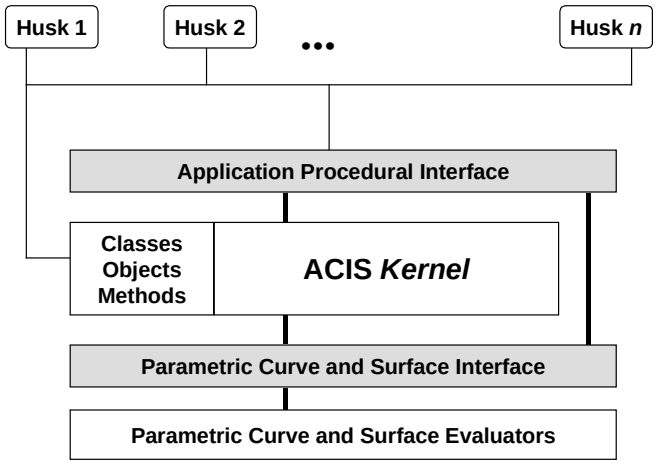


Figure 10 - ACIS architecture

(3) a standard geometric data format.

ACIS provides these items through the ACIS architecture, ACIS components (called *Husks*) and ACIS SAT file format. The architecture of the ACIS Kernel Modeller is shown in *Figure 10*. ACIS Kernel is a Boundary-representation (B-Rep) modeller that integrates 3D solids, surfaces and wireframes. A Geometry Bus, as defined above, requires the reuse of software components. ACIS meets this requirement through the use of component modules called *Husks*. In a lower level, the ACIS C++ class structure is another platform for the reuse of software components. Any of these components (lower-level modules or *Husks*) can be substituted by a newer component developed by the programmer without jeopardising consistency and interoperability.

ACIS files can be browsed through the internet by freeware browsers, what is expect to happen when a component is stored in the Product Structure. However, during the design process, solids should be modified and not only browsed. In the proposed distributed system, ACIS solids are loaded in the working memory by an operation that is stored in database tables. *Figure 11* illustrates how an entity called Piston Rod is represented and built using ACIS's methods and objects. The entity Piston Rod has a Solid that belongs to the class **Cylinder**. The operation **AcisMake Cylinder** in *Figure 11* is stored in a database table, which might be in a remote computer. ACIS operations are always considered attributes of the object Topological. If no values of *radius*, *height* and *position* are available, the predicate *if-needed* attached to the attribute **AcisMakeCylinder** will be fired and appropriate methods will provide values for the parameters required. Any change in these values will cause the predicate *if-changed* to be fired, keeping the data and the visualization updated on the entire network.

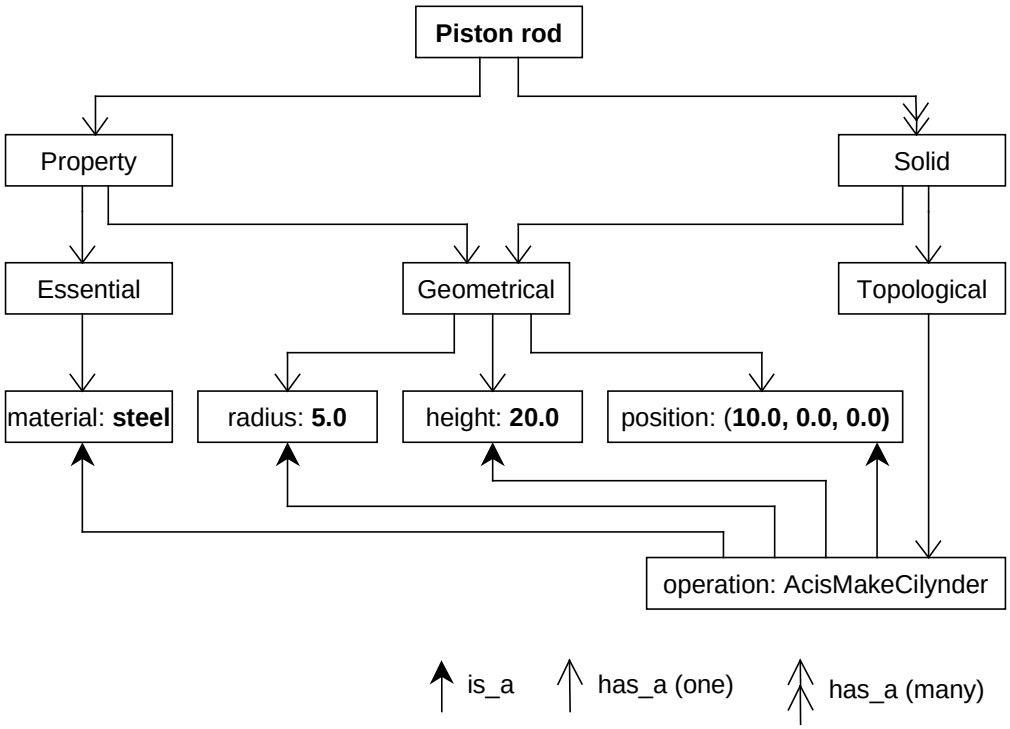


Figure 11 - Geometrical operations

Conclusions

The highly dynamic aspects of the design process in a collaborative CAD environment require the use of dynamic distributed objects. There are few references in the literature on this matter and most of the published work presents architectures based on the static interfaces of CORBA. This paper presents a distributed dynamic data model that is aligned with the concepts of design as problem solving and design history. In the present paper, the dynamic aspects of the distributed objects are provided by a mechanism that is not dependent on CORBA's DII (Dynamic Invocation Interface). Furthermore, this paper presents a solution to the problem of having data distributed in a heterogeneous network with several different databases.

In this paper there is no intention to present a formal framework that integrates workflow(Winograd, 1994) (Georgakopoulos et al., 1995) (Alonso et al., 1996), planning (Kerzner, 1997) and traceability (Gotel and Finkelstein, 1994) (Ramesh, 1995) aspects, nor the authors intend to propose a distributed and collaborative transaction model for databases (Alonso et al., 1996). Also no attempt is made to tackle problems of data consistency, version control, workflow control and documentation control. The model is not a framework for Object Oriented Heterogeneous Data Base Management System (Uchôa, 1999), but it is a basic foundation to build an integrated framework in future, which is capable of considering most of those aspects found in collaborative CAD environments.

Several concepts of the present paper have been used to develop a CAD working system for the industry (Barbosa, 1996) (Barbosa, 1997a) (Barbosa, 1997b) and there is also an academic prototype, based on this model, already performing some tests.

Acknowledgements

The authors would like to acknowledge CNPq, Brazil, and FCT, Portugal, for their financial support. The authors would also like to thank Embraer, the Brazilian aeronautics company, and Petrobras, the Brazilian oil company, for making possible the development of an academic prototype and a CAD working system using the concepts of the present work.

References

Almasi,G., Suvaiala,A., Goina,C., Cascaval,C., Jagannathan,V.; 1999; TclDii: a TCL interface to the Orbix Dynamic Invocation Interface; http://www.cerc.wvu.edu/iss/TclDii.htm .

Alonso,G., Agrawal,D., El Abbadi,A., Kamath,M., Günthör,R., Mohan,C.; 1996; Advanced Transaction Models in Workflow Contexts; *Proceedings of the 12th Int. Conf. On Data Engineering, Feb.26-Mar.1, 574-581*; IEEE Computer Society; New Orleans, Louisiana; USA. [Also in http://www.almaden.ibm.com/cs/exotica/exotica_tran_models0795.ps].

Barbosa,C.A.M.; 1996; Um Modelo para a Representação do Aspecto Evolutivo do Processo de *Design*; Dissertação de Mestrado, Dpto. de Engenharia Mecânica, PUC-Rio, Brasil.

Barbosa,C.A.M.; 1997a; GerLay – Manual do Sistema; Petrobrás/CENPES; Rio de Janeiro, Brasil.

Barbosa,C.A.M.; 1997b; GerLay – Manual do Usuário; Petrobrás/CENPES; Rio de Janeiro, Brasil.

Bento,J.; 1992; Intelligent CAD in Structural Steel: A Cognitive Approach; PhD Thesis; Dept. of Civil Engineering, Imperial College of Science, Technology and Medicine; London, UK.

Feijó,B., Gomes,P.C.R., Bento,J., Scheer,S., Cerqueira,R.; 1998; Distributed agents supporting event-driven design processes; *Artificial Intelligence in Design'98, Gero,J.S., Sudweeks,F. (eds), 557-577*; Kluwer Academic Publ.; The Netherlands.

Georgakopoulos,D., Hornick,M., Sheth,A.; 1995; An overview of workflow management: from process modeling to workflow automation infrastructure; *Distributed and Parallel Databases, 3(2), 119-153*; Kluwer Academic Publ.; The Netherlands.

Gotel,O.C.Z., Finkelstein,C.W.; 1994; An analysis of the requirements traceability problem; *Proceedings of First Int. Conf. On Requirements Engineering, April 18-22, 94-102*; Colorado Springs, Co.; IEEE Computer Society Press; Los Alamitos, CA, USA.

Houlding,D.; 1998; A CORBA bean framework; *Dr. Dobb's Journal, November, 23(11), 34-40*; San Francisco, CA, USA.

Ierusalimschy,R., Figueiredo,L.H., Celes,W.F.; 1996; Lua – an extensible extension language; *Software: Practice & Experience, 26(6), 635-652*; John Wiley & Sons; New York, NY, USA [language available in http://www.tecgraf.puc-rio.br/lua].

Inference Corporation; 1985; ART Programming Manual; Inference Corporation; Los Angeles, CA, USA.

Kerzner,H.; 1997; Project Management: a Systems Approach to Planning, Scheduling, and Controlling; 6th edn., John Wiley & Sons; New York, NY, USA.

Levitt,R.E., Christiansen,T.R., Cohen,G.P., Jin,Y., Kunz,J.C., Nass,C.I.; 1994; The Virtual Design Team: A Computational Simulation Model of Project Organizations; *CIFE Working Paper 29*; Stanford University; Stanford, CA.

Maher,M.L.; 1990; Process models of design synthesis; *AI Magazine, Winter, 49-58*; Menlo Park, CA, USA.

Minsky,M.L.; 1988; The Society of Mind; Pan Books, London, UK.

Ramesh,B., Powers,T., Stubbs,C.; 1995; Implementing requirements traceability: a case study; *Proceedings of 2nd IEEE Int. Symposium on Requirements Engineering, March 27-29, 89-95*; IEEE Computer Society Press; Los Alamitos, CA.

Simithers,T., Troxell,W.; 1990; Design is intelligent behaviour, but what's the formalism?, *AI EDAM, 4(2), 889-898*; Cambridge University Press; MA, USA.

Simon,H.A.; 1969; The Science of the Artificial; MIT Press; Cambridge, MA, USA.

Spatial Technology, 1998; ACIS 3D Toolkit 4.0; Spatial Technology; Boulder, Colorado, USA.

Uchôa,E.M.A.; 1999; Framework para Integração de Sistemas de Bancos de Dados Heterogêneos; Tese de Doutorado, Dpto. de Informática, PUC-Rio, Brasil.

Widom,J., Ceri,S.; 1995; Active Database Systems: Triggers and Rules for Advanced Database Processing; Morgan-Kauffman, Inc.; San Francisco, CA, USA.

Winograd,T.; 1994; Categories, disciplines, and social coordination; *Journal of Computer-Supported Cooperative Work, 2(3), 191-198*; Kluwer Academic Pub., The Netherlands.