A framework for a distributed CAD system

Laurent Denis a,⁎, Yvon Gardan a, Estelle Perrin b

a CMCAO team, CAD Computer Science Lab. IFTS, Pôle de Haute Technologie, 08000 Charleville-Mézières, France
b CMCAO team, CAD Computer Science Lab. Metz University, UFR IMM, Île du Saulcy, 57045 Metz Cedex 01, France

Accepted 12 September 2003

Abstract

The DIJA project is a web-based CAD system accessible to any user (from neophyte to expert) and from a simple desktop computer. Its modeling approach is based on a top-down methodology and takes into account trade information. In this paper, we propose a network architecture based on a replication process and a multi-level language to manage the consistency between the client model and the server model. The whole CAD system is based on a modular architecture communicating with messages exchanges and provides a general architecture for all trade CAD applications.

Keywords: Distributed architecture; Web-based software; Modeling language

1. Introduction

More and more commercial companies try to enhance their CAD systems to take advantages of web-possibilities probably because of financial interest. But this goal is confronted with several problems:

– As any client computer has to load software and data on the server computer, communications between these two computers have to be fast enough. Then the choice of the system architecture and the model is essential. Unfortunately, classical models (such as B-Rep and CSG) used in CAD systems are inappropriate: they are relevant for the goal they have been created: geometry;

– Moreover, in an internet context, the user skills change: instead of being always an expert, it may be a neophyte. So, CAD systems have to improve their interface in using semantic notions. The interface, and the tools proposed have to be clear enough to be used by a user who has never heard talking about vertices, Boolean operations,…, but who is an expert in his trade (for example, foundry, metalworking industry,…).

The main objective of the DIJA project is then to define and to implement an adapted architecture for such a web-based CAD system, to create a system for any user (from neophyte to expert) and accessible from a simple desktop computer. To provide the user with an easy-to-use system we have already proposed a new design method called ‘synthetic approach’ [1] coupled with an intuitive interface. The both are based on visual entities named dialog elements which are enough to totally define and manipulate a design [2].

Briefly, the synthetic method is a deductive approach for the design of CAD parts: the user starts from a shape which represents the global idea of his part. This shape can be chosen in a library or the user may create it from nothing. With simple and dedicated tools, he refines it to obtain his final part. These tools may be totally free of semantic or more functional because linked to a trade. So, in DIJA, each trade domain has its proper rules and vocabulary to describe functional tools. The language of the DIJA interface is varying according to user’s trade. At any step, the user can choose a high or low semantic tool and the interface language evolves with his choices. For example, if the user wants to design a pitcher, he can describe it first as a container. Such a word suggests a lot the design (and the initial shape) but it does not completely define it.

The DIJA project allows different users to design the same part. These users can be located at different places but the modifications have to be asynchronous. This kind of applications is particularly intended to be used in an extended enterprise context. In such a context, different
people from different trade have to work on the same part: for example, the prime contractor defines indeed the general characteristics of a part and a subcontractor refines some of these characteristics in order to obtain a specialized view of the final part. So it is important to offer the user an adapted semantic to describe his part. In the previous example, the prime contractor uses general terms to define the part whereas the subcontractor needs a more specialized language.

In this paper we want to focus on the distributed architecture of the DIJA project. When a client wants to connect to a server in order to realize a design, the two computers have to share a lot of data: whatever the quantity would be, the client does not have to wait a long time before working. In the next sections, we show how to distribute CAD models (see Section 2), how to apply compression methods to these models and how to send partially a model in using incremental refinement methods. We try to adapt them to CAD systems and show their limits in the context of the DIJA project. We then show the model used in the DIJA project: a multi level language that guarantees to realize client-server communications (see Section 3). Based on the language, the whole CAD system is based on a modular architecture communicating with messages exchanges and provides a general architecture for all trade CAD applications (see Section 4). We finally propose a network architecture based on a replication process to manage the consistency between the client model and the server model (see Section 5). We end this paper by showing one of the applications implemented with the DIJA project (see Section 6).

2. Previous work

Distributed environments have to maintain consistency among shared data stored on several computers. As a result, whenever an entity modifies the shared environment, an appropriate update must be applied on every copy. In the next sections, we study some distributed systems and we present some techniques dedicated to geometric data exchange.

2.1. Distributed systems

In a distributed CAD system, a client has to use data stored on a server. The simplest way to realize this is to store data only once on the server and to redirect any access to these data via remote procedure calls. Remote procedure calls provide a solution to manipulate objects from a distant computer. They use the notion of client stub and server stub. When the program on the client computer calls a method on a remote object, a request is sent to the client stub which dialogs with the server stub, by transferring the call and the arguments. The server stub uses them to call the real method before returning the result by the reverse way. Several tools like DCOM (Distributed Component Object), CORBA (Common Object Broker Architecture) or Java RMI (Java Remote Method Invocation) implement remote procedure calls. New improvements in this field allow to use such technologies in real time applications like medical software framework [32]. However, these applications use high speed network to exchange data. In our CAD system, we wish to use a standard network connection (like DSL technology). So data access has to be fast enough in order to provide the user with a feedback. Thus it is not suitable to use RPC solutions because the network bandwidth does not allow such a feedback. As RPC solutions are not appropriate [3], we have to store data locally on the workstation. Another way of sharing data is to provide the client with a replica of the data and to ensure the consistency between the two models. Replication process consists in copying on the client the data that the remote program needs among those stored on the server, and in ensuring the consistency at each modification realized on a data or on one of its replicas. Such an approach is used in several applications that provide the user with a distributed environment or a collaborative work environment:

- The distributed implementation of Open Inventor [4] which is a high level 3D graphics API serializes events generated by users. Changes on a scene are distributed via reliable multicasting;
- Ring [5] takes advantage of the fact that state changes must be propagated only to hosts perceiving changes;
- Repo-3D [6] uses a sharing mechanism based on a distributed shared memory. This mechanism allows a computer to be programmed like a multi-processor;
- Silveira and Ting [7] use an hybrid approach. A remote part is dedicated for the geometrical data and a replicated part for 3D graphics;
- In their CAD system, Bajaj et al. [8] use geometric modeling modules based on a geometric programming language called PLaSM [9] and a java enhanced VRML browser to visualize the scenes. PLaSM is a functional oriented geometry language which describes geometry. Visualization is done with a dedicated viewer.

As all these applications mentioned above, the DIJA project uses a replicated approach. But, in order to provide the user with a good reactivity, exchanges have to be the most compact as possible. So to minimize the size of exchanged data, compression and incremental methods adapted to geometric exchanges are studied in the next section.

2.2. Compression methods

Compression methods provide a more compact representation of an object with no loss of information. The most common representation used in graphics is meshes. A mesh
is represented by topology, geometry and properties. The topology is the incidence relation among the vertices. Geometric data describe the location of each individual vertex. Properties are additional data typically used to describe mesh appearance (normal, color, texture...). Usually, all these information is compressed in a separate way. In this paper, we only focus on topology and geometry.

Graphics APIs like OpenGL use the notion of triangles strip to reduce the amount of memory used to represent topology. A mesh is decomposed into a series of strips. A strip is only represented by a list of vertices. Taubin and Rossignac [10] use a similar approach. They create a vertex spanning tree by selecting the minimum number of edges that connect all the vertices without creating loop. The mesh is cut through these edges to obtain a union of triangle strips. This algorithm encodes connectivity information in an average of 4 bits per vertex. Rossignac [11] proposes an approach called edgebreaker based on a state machine. Edgebreaker uses a recursive procedure which traverses the mesh from one triangle to a neighboring one, classifying each triangle with a set of five symbols which are stored in a list. The classification and the choice of the next triangle are made according to the adjacent elements which have been previously visited. Edgebreaker requires at most 2 bits per triangles for any mesh homeomorphic to a sphere. Touma and Gotsman [12] propose to compress a triangular mesh using the valence of every vertex (number of emanating edges). The encoding algorithm starts with an arbitrary triangle defining an active list of three vertices. The first vertex of the list is called the ‘focus’. For each edge \((v_{\text{focus}}, v_{\text{new}})\) emanating from the focus, the algorithm stores the new vertex \(v_{\text{new}}\) and its valence in the active list. When all edges are exhausted, the focus passes to the next vertex in the active list. If the active list self-intersects, it is split into two active lists. The obtained active list is the compressed representation. This method encodes connectivity information in an average of 1.5 bits per vertex. The compression ratio given depends on the mesh connectivity and its number of triangles. The higher the number of triangles is, the better the compression is.

To complete the encoding, the previous methods have to be combined with geometry compression methods. Generally, geometry is compressed using prediction methods. The encoding of the coordinates of a vertex is function of the encoding of the previous ones: the position of the next vertex is predicted according to the coordinates of previous vertices and only the error between the real position and the prediction is encoded. The performance of geometry compression depends on the prediction accuracy for vertex position. One of the simplest methods is linear prediction. The position of the vertex \(v_n\) is supposed to be aligned with two previous vertices \(v_{n-1}\) and \(v_{n-2}\). Moreover, the length of \([v_{n-1}, v_{n-2}]\) is supposed to have the same length of \([v_{n-1}, v_n]\). Another simple method is to lay out the vertices in order to create a parallelogram. This method requires the position of the three previous vertices \(v_{n-1}, v_{n-2}\) and \(v_{n-3}\).

2.3. Incremental methods

Incremental methods start from a coarse shape and proceed by refinement. Several methods use models like octree representation or subdivision surface which allow directly such a refinement. Other methods are based on more traditional models like meshes and have to store an initial shape and a set of modifications used for the refinement process. These modifications are computed from simplification techniques applied to the original object [13,14].

Taubin [15] reconstructs a uniform subdivision Loop structure [16] from irregular triangle meshes. Brunet et al. [17] improve classical octree representation by creating new octree nodes that contains parts of the surface of the object. With such a representation, they are able to obtain an exact representation of polyhedral object. To progressively send the representation, Cano and Torres [18] include some additional information in the internal nodes in order to obtain good approximation in a few iterations.

When using meshes, lists of modifications are generally obtained by a simplification method based on decimation. These methods work by iterative elimination of elements (vertex, edge, face). Usually, the algorithm determines a weight for all elements. This weight represents the importance of the element according to the criterion the user wants to optimize (curvature, volume, etc.). Then, the less important elements are eliminated and the weight of other elements is updated. Cohen et al. [19] compute two envelops at a certain distance of the object and eliminate vertices until the object does not intersect with envelops. Lindstrom and Turk [20] remove edges according to a volume criterion. This algorithm does not keep information from previous steps, so it is memory efficient and adapted to very large polygonal models. Taubin et al. [21] remove triangles strips which are stored in a progressive forest split format. This format efficiency stores removed triangles and allows progressive transmission. Hoppe [22] with progressive meshes uses random edges decimation to simplify a mesh. Cignoni et al. [13] have tested several of these algorithms on an SGI Indigo2, R4400 200 MHz CPU. Dividing by 2 the number of triangles of a 70,000 triangles mesh requires time from 5 to 50 seconds depending on the algorithms.

2.4. Conclusion

Most of these methods use meshes as input and output. In order to use the methods presented in this section, Masuda and Ohbuchi [23] translate CAD models into a mesh. But CAD systems use more complex models that may contain surfaces, solid forms and various non-simple topological structures like cavity, features or constraints assembly information. Such entities are lost when translating into meshes. Danesi et al. [24] notice that dedicated methods to rapid data transfer do not concern CAD models. So they suggest to use a functional model to naturally compress...
data. But deducing a geometric model from a functional model is still an open problem.

The DIJA project proposes a deductive approach to solve this problem: at the very first time of a design, geometry does not exist. The system deals first with specifications. Then geometry is deduced as the design progresses. Specifications information is very compact; therefore it can be used for fast data exchanges. In the next section, we present the model we created to realize this refinement.

3. Functional model

The DIJA project is indeed based on a top-down approach where the user starts with a global idea and refines it to obtain his final part. During the refinement phases, the user specifies his part with a more and more precise vocabulary. So, the model used to describe actions made by the user is a language (a set of instructions explained in Section 3.2) that takes into account this methodology by proposing several levels of abstraction. Five levels of abstractions are used in the conception: application, trade, common, dialogs and geometry (see Section 3.1).

On top of that, a language is an efficient way to exchange data between a client and a server. It provides indeed a high level of abstraction and a compact representation of geometry. The levels are also used to find an adequate level of abstraction and a compact representation of data between two computers, the system used instructions to gather several trades to make a new application. This process is named refinement phase. The parameters are instructions names. In fact, a parameter indicates the set of data the corresponding language involves numerous calculations to obtain geometric elements but low transfer time. On the contrary, low level of abstraction language involves huge data volume to send but high transfer time. In order to have optimal request time we have to find an adequate level of abstraction of the language.

3.1. A multi-level framework

During the design, the part is viewed at different levels of abstraction. Each level of abstraction differs from the others from the vocabulary of the language they use. The different levels of abstraction are from the highest to the lowest:

- **application level** is the highest level of abstraction. It deals with the different elements needed to create a specific application. For example, an application concerning building architecture could manage data belonging to masonry, carpentry… So this level is able to gather several trades to make a new application. This application level also uses language instructions concerning system behavior, metascript, etc.

- **trade level** is the level where all language instructions are associated with a particular trade. For example, ‘to hollow out a part’ has different meanings according to the context used. In the case of plastic processing, ‘to hollow out’ means to withdraw material and to create a network of ribs in order to increase the rigidity of the part, whereas in the case of machining, it means only to withdraw material. It is then relatively easy to express, for each trade, specific concepts without changing the main model. Furthermore, the trade level allows the CAD system to present a design and its associated know how from several views;

  - **Common level** is an intermediate level which uses trade general concepts. For example, the trade notion of mortise is a notch with a particular geometry;

  - **Dialog level** is the level where all language instructions deal with dialog elements [2]. We recall that dialog elements allow modifications on the geometry of the part;

  - **Geometrical level** is the lowest level language used to create specialized models. The language is indeed relatively abstract and it is necessary to translate it into specialized models (meshes, boundary representation, etc.) to be able to make tasks like visualization or simulation. The intrinsic vocabulary of this level is based on the names of the primitives used to access to the information present in specialized models. This level uses a specific solid modeling system.

3.2. Multi-level language

In order to describe design information and to exchange data between two computers, the system used instructions belonging to a multi-level language. Whatever the level is, each instruction is basically composed by four fields named ‘name’, ‘action’, ‘parameters’ and ‘evaluation’. The name field makes it possible to identify an instruction in a unique way. The meaning of the instruction is contained in the action field. The action field enables the CAD system to understand the instruction and to refine it into a corresponding set of instructions of the same level or of the lower level or in the local host model language. This process is named refinement phase. The parameters are instructions names. In fact, a parameter indicates the set of data the corresponding instruction has created or modified. For example, if ‘instruction 1’ creates a cylinder and ‘instruction 2’ creates a hole with a parameter whose value is ‘instruction 1’, this means that “instruction 2” makes a hole in the previously created cylinder. The evaluation field is typically used when the CAD system needs to receive some information from the user.

If the system has to obtain the geometry of the design, it translates any level instructions until it reaches the geometrical level. The lowest level language is used to create specialized models suited to the task the user wants to do. If the client needs to display and manipulate geometry, its model could be for example a boundary representation. On the contrary, if it wants to make simulation, its model could be a model for finite elements. Thus, the use of the language can be seen as a common interface with different
translations according to the underlying model. So the language bridges the gap between several models. In the next section, we explain how the system manages instructions in order to get the geometry.

4. System design

As the DIJA project provides the user with a high semantic level language, a word of this language has to be refined until it becomes one or several instructions of a classical geometrical model. Such a process is not instantaneous and is realized in the DIJA project by a modules hierarchy that is described in this section. The architecture [26] takes up the five abstraction levels mentioned in Section 3.2. Each level of abstraction is made up of several modules and is call environment (Section 4.1). One level only communicates with its immediate lower level (Section 4.2). The DIJA project can be seen as a general framework for web-based CAD systems [27], because it provides a general architecture for all trade CAD applications.

4.1. A modular environment

An environment is made up of modules designed to make specific tasks. In the DIJA project, we distinguish five modules:

- **the executive backbone module (EBM)** receives all instructions of the language. It gives all instructions to the historic module to save them. If it does not understand an instruction, it sends it to the knowledge base module, otherwise it executes it or send it to the appropriate module (EBM of the lower environment or CHIM). These modules can be located in the same computer or on the remote computer. If they are on the remote computer, the EBM uses the distributed architecture described in Section 5;
- **the knowledge base module (KBM)** is a module which contains the know-how of the design. When the KBM receives an instruction from the EBM, it returns a script which explains how to realize the instruction;
- **the computer human interaction module (CHIM)** receives instructions from the EBM. It allows the user to interact with the CAD system. According to the received instruction, it can ask the user to give some values, to draw a dialog element, etc. The result of the interaction is also an instruction eventually empty which is sent to the EBM;
- **the visualization module (VM)** allows the user to see the result of its design at the CHIM request;
- **the historic module (HM)** is the model used to store the design. It receives all instructions from the EBM and saves them. It is also able to answer to requests sent by other modules via the EBM. After a working session, client data are updated in order to allow another client to work with the new information.

A same module may have several implementations, and several implementations of a same module can be used on several clients working on the same project. Each of these implementations provides the user with a specific view. For example, the visualization module may differ according to the trade of the user. Thus, for a same part, it is possible to focus on the geometry or on functional information.

Fig. 1 represents the relations between the different modules. The arrows represent a communication. The direction of the arrow shows the way of the communication. Even if an instruction may induce an answer, no reverse arrows are drawn between modules.

Fig. 1 includes all modules (for one environment) which can be used on each computer. However, all these modules do not need to be loaded on a computer. For example, in the case of an extended enterprise, the prime contractor puts a server at subcontractor’s disposal. Then the two enterprises work on the same model. In this case, the knowledge base module appears both on the client and the server. But, as the subcontractor may not want to share his know how, the DIJA project allows the data stored on the client and the server to be different.

Then only the historic and the executive backbone modules have to be on each computer because they are always used during the communication process. Therefore, in order to avoid downloading the classes and the data required each time the user logs on, the different modules and their data are stored locally. Data and the choice of modules can be adapted according to the enterprise needs. A classic configuration can be seen in Fig. 2.
4.2. Communication

A summary of the role of each module is presented in Section 4.1. Here, we describe more precisely how the EBM (executive backbone module) works. This module has two distinct roles. On one hand, it translates script provided by the KBM (knowledge base module); on the other hand it sends the obtained instructions to the other modules of the same level or to the EBM of the immediate lower level and waits for a result.

An instruction describes a task. If the EBM does not understand the instruction (for example, it is not in a native language), the instruction needs to be translated by the KBM. This module is able to provide the system with a general way to solve the problem described by the instruction. The given solution is a script. A script is a succession of tasks expressed as an algorithm. To provide the system with a set of instructions to execute, the script has to be translated. The translation is made by the EBM. The result of the translation is a set of instructions which has to be sent to the different modules. A design is then described by all these instructions, so they are stored in the historic as and when they are generated.

The communication between modules is based on instructions exchanges. When a module receives an instruction from the EBM, it treats it and returns a response to the EBM. The result can be the evaluation of an instruction or a new instruction generated by the module. In the case of evaluation, the sent instruction and the returned instruction are the same. Only the evaluation field changes and is completed. In the other case, the returned instruction is a new instruction which has to be translated by the EBM. All these actions are summarized in Algorithm 1. All communications are then overseen by the EBM and the different modules only communicate through the EBM. This means that all modules use the EBM to send and receive messages.

Algorithm 1 assumes that the KBM is always able to translate an instruction into a script. But this is not always the case. So, when an action cannot be performed or information cannot be found locally, the local computer uses the remote computer (see Fig. 3). In this case, the request is sent to the server EBM of the same level. The server treats the instruction and returns the result. The design continues as if this result was given by the local computer. The next section explains how data exchanges are made between a client and the server.

5. Distributed architecture

We then first recall that the DIJA project uses a replication process in order to provide the user with the capability to deform interactively his part and to view the result in real time. This process is realized with a language (see Section 3) on which compression and incremental methods are unusable (see Section 2). When our system is used in a distributed environment, the server has a central role. Each client has only to communicate with the server and no communication between clients is allowed. As seen in Section 3.2, communications between the server and the client are made with the language, which lets their
geometrical models to be different. The server saves the whole design (see Section 4.1) and the client loads only desired data. Then specific communications have to be implemented for integrity management. In the next sections, we explain how the server and the client update their models (5.1) using language instructions and how such instructions are sent through the net (5.2) in order to realize the distributed system.

5.1. Data exchange

In the executive backbone module (see Section 4.1), an instruction $i$ of one level of abstraction produces several other instructions $i_1, \ldots, i_n$ of a lower level of abstraction in one or several iterations. The instruction of level $i$ represents the conception at one level of abstraction while the instructions $i_1, \ldots, i_n$ represent the same conception at another level of abstraction. For example, a hole that can be represented by the action instruction field ‘to make a hole’ will refine into several instructions whose actions are ‘cut face’, ‘create vertex’, ‘create edge’ and ‘create face’. Now, if we want to modify an instruction, the number of sub-instructions generated by this instruction and the action they have to realize may change. If the number of instructions increases, their names must not interfere with the names given to other instructions. So we propose to name the instructions hierarchically. As five levels of abstraction are manipulated, each instruction is named with five integers $(\text{level}_5, \text{level}_4, \text{level}_3, \text{level}_2, \text{level}_1)$. For example, the third sub-instruction of level 4 created by the fourth instruction of level 5 is called $(4,3,0,0,0)$. So the names of all sub-instructions created by an instruction start by the same numbers.

Such an instruction can either be sent or received by the client or by the server. So to avoid conflicting names, a naming system gives to the server and the client the appropriate integer for each level. This naming system is necessary as soon as there is a communication between the client and the server. Because of replication process, this communication may concern the update of shared data. But specific treatments may be computed on the server at the client request because the client is too slow or because treatments concern know-how the server wants to keep.

In the next paragraphs, we show that our naming system is relevant in studying the only three following possible cases: the client loads data from the server, the client updates data on the server, and the client requests the server to do some treatments. These cases are illustrated with the formalism presented in Fig. 4.

5.1.1. Client loads data from the server

At the beginning of a session, the client needs to receive its data from the server (Fig. 5a). In order to start working before the reception of all data, the server sends to the client the first name of instruction it can use (Fig. 5b). Then the client starts to generate instructions from this first name (Fig. 5c).

If the client can refine an instruction (according to its resources, the instructions already received and its know-how), the server does not have to send the corresponding sub-instructions. Such sub-instructions are marked as updated on the server even if they have not been sent through the network.

5.1.2. The client updates data on the server

To update the server, the client has to send all instructions which have been created or modified since the last update (see Figs. 6a and b). In order to minimize the exchanges, all sub-instructions that the server is able to refine by itself are not sent and are considered as updated.

5.1.3. The client requests the server to do some treatments

When the client requests the server to do some treatments, the server has to update its data (Fig. 7a).
The request may include calculation like simulation which
not modify the model and treatments which modify the
model. In the first case, the system allows the client to
generate and to modify instructions. But the server answer
will concern the model in a previous state. On user request,
the system will be able to open a new session using the state
responding to the treatments demand. In the second case,
the server may modify or add some instructions shared with
the client. Then the ones on the client are not up-to-date. So
during the same time, if the client uses them, the design is not
consistent. A solution is to lock such instructions. But the
server may modify an instruction a long time after the
beginning of the request and the client may modify the same
instruction just after the request, so we cannot lock client
instructions just after a server modification. Furthermore, it is
not possible to know exactly which instructions will be
modified when the client requests the server. Then the system
has to find all possible instructions that could be modified to
lock them. The locked instructions are then a upper set of
modified instructions. To solve this problem, the system
creates an oriented graph whose nodes are client instruc-
tions [25]. An oriented arc is added between instruction
\(i_1\) and instruction \(i_2\) if instruction \(i_2\) is a sub-instruction of
instruction \(i_1\) generated during the refinement phase or if
instruction \(i_1\) is one of the parameters of instruction \(i_2\).
Then two instructions \(i_1\) and \(i_2\) are indirectly linked if an oriented
path from \(i_1\) to \(i_2\) can be found. The graph is then covered
from the instruction corresponding to the treatment the server
has to compute until leaves are reached. Each met instruction
is a locked instruction (see Fig. 7b). When the request is
finished, the system has to update the client model (Fig. 7e).

With this configuration, the server and the client can
generate instructions at the same time (see Fig. 7c). Then
client instructions names may conflict with instructions
names added by the server. The system is unable to know the
number of instructions created by the server, so it temporarily
names the new client instructions from ‘0,0,0,0,0’ (see
Fig. 7b) and renames them (when the server has finished to
compute) with the algorithm described below:

- the system finds the last name \(n\) generated by the server,
  for example ‘2.2.3.2.7’;
- the system replaces the \(d – 1\) last integer of \(n\) by 0 to
  obtain the translation vector. The translation vector
  associated with the example is ‘2.2.3.2.0’;
- finally, the system adds the translation vector to all
  temporary names and references (Fig. 7d). Then name
  ‘0.0.0.38.4’ becomes ‘2.2.3.40.4’.

In conclusion, the naming mechanism is validated in
each case. It ensures the naming of instructions with a
unique name, and encapsulates the dual creation of
instructions on both computers.

5.2. Data stream management

When data are sent through a network, several problems
may occur in addition to network rate or naming system:

- the communication may be broken between the two
  computers;
- data could be picked up or hacked.

Then the network architecture has to manage the
exchanges between the client and the server in taking into
account these problems. So, the network architecture is
organized in layers. Each layer is implemented as a module,
providing the system with a specific task. Five layers
compose the architecture scheme:

- network module: this module concerns the entire
  naming system explained in Section 5.1;
- correspondence module: this module is able to know
  which data have been updated on both computers;

Client

Server

Client

Server

---

(a) 

(b) 

---

Fig. 5. The client loads data from the server.

Fig. 6. The client updates data on server.
– security module: its role consists in securing the exchanges by encrypting data which are going to be sent;
– compression module: this module is not a compression module as those described in Section 2. But even if the choice of the abstraction level allows us to determine the amount of data exchanged and if necessary, more compact data can be sent in using usual compression algorithm, like Lampel-Ziv compression;
– buffer module: the client and the server do not have to be always connected. So, in this case data are temporally stored in a buffer. Using a buffer allows the system to have the same behavior even if the connection is broken. When the connection is set, the buffer is emptied.

Each computer has the five modules as shown seen on Fig. 8. When a module makes an action on the client computer, the same module makes the reverse action on the server computer. For example, if the client security module encrypts information, the server security module deciphers it.

The network architecture presented above allows the system to exchange data between two computers in a replication process context. As explained in Section 3.2, in order to improve the amount of information exchanged, we use a multi-level language. The different semantic levels allow the system to adapt the amount of data exchanged according to the network bandwidth and the remote computer capabilities. Furthermore, using such a language allows the client and the server to use different geometrical models. Thanks to the naming system, a task can be computed equally on the client or on the server. Finally we have completed the architecture by adding modules which deal with tasks like encryption and compression.

6. Implementation

All abstraction levels described in this paper have been implemented using the SDK 1.3 of the Java language and the Java 3D library for visualization. Two applications have been developed to validate the architecture of the DIJA project in using these levels. The first one is dedicated to the art of entertainment, the second one to foundry. One of the main problems of any founder is the design of the filling system. When a founder designs a filling system, he uses explicit and implicit knowledge [28]. Thanks to this

![Fig. 7. The client request the server to do some treatments.](image1)

![Fig. 8. Network architecture.](image2)
knowledge, they calculate and model the filling system that is totally defined by a cup, a downsprue, a runner and a riser (see Fig. 9). The role of the cup is to receive molten metal and to direct it towards the downsprue. Then metal goes into runners and then into ingates (which are the links between the filling system and the part) before filling mould cavity which represents the part. The riser is a molten metal tank that regulates the problem of the matter withdrawals during solidification.

When the user runs the DIJA software, he first has to choose the trade of the application (in our case foundry). In the system, this interaction is translated into a receipt of initialization information to the executive backbone module (EBM). The EBM asks the knowledge base module (KBM) to obtain a set of available applications. The EBM sends this information to the computer human interaction module (CHIM) that organizes displays and proposes a choice window. Then the user can start the design of a filling system. It is created thanks to a script given by the KBM. This script describes the following different stages to obtain the filling system:

- the CAD system asks the user to give the trade parameters (including the plate model size that leads to the size of the whole filling system (part in gray in Fig. 11)) of his filling system. This step is made thanks to the computer human interaction module (see Fig. 10);
- the CAD system loads the part. On the part, the user has to locate the ingate(s) and the riser(s);
- the CAD system computes the size of the different filling system components thanks to the trade parameters given in step 1 (especially the plate model) and thanks to parameters from the knowledge base module.
- the CAD system creates the filling system according to the previous computations (see Fig. 11);
- in order to verify that the system is correct, the CAD system asks a simulation software to simulate the molding process. As this task required numerous calculations, it is made on the server with a dedicated software. The filling system creation process is reiterated until the results are satisfactory.

All instructions needed to implement the script are stored in a XML (eXtensible Markup Language) format. The reason is that the data structure exchanged between modules may involve as designs go along. Several fields are indeed used only as and when they are need (for example, the evaluation field). Moreover, the type of the different fields used to describe an instruction may rapidly involve. That is the reason why these data can be viewed as semi-structured data [30]. Since 1998, XML formalism is more and more used to represent such data [31]. XML is a markup language used on the web to
structure a document and in applications to store or to exchange data. Furthermore, XML provides the programmer with well normalized APIs implemented in numerous programming languages.

The instructions corresponding to the script for creating the filling system are shown in Fig. 12. Each of these instructions may be refined into same level or lower level instructions. For example, the instruction ‘compute size’ is refined into several instructions which compute the size, the position, the orientation, and the number of the different components of the filling system (runner, ingate…) (see Fig. 13).

By refining all the instructions, the system finally obtains a geometrical representation of the filling system (see Fig. 11). The dependency graph explained in Section 5.1 is partially represented in Fig. 14.

```
<instruction>
  <name> 6.6.0.0.0 </name>
  <action> choose parameters </action>
</instruction>
<instruction>
  <name> 6.7.0.0.0 </name>
  <action> load part </action>
</instruction>
<instruction>
  <name> 6.8.0.0.0 </name>
  <action> compute size </action>
  <parameter> 6.6.0.0.0 </parameter>
  <parameter> 6.7.0.0.0 </parameter>
</instruction>
<instruction>
  <name> 6.9.0.0.0 </name>
  <action> create system </action>
  <parameter> 6.8.0.0.0 </parameter>
</instruction>
<instruction>
  <name> 6.10.0.0.0 </name>
  <action> simulate </action>
  <parameter> 6.9.0.0.0 </parameter>
</instruction>
```

Fig. 12. Set of instructions obtained after executing the script “compute a filling system”.

```
<instruction>
  <name> 6.11.0.0.0 </name>
  <action> compute downsprue size </action>
  <action> compute cup size </action>
  <parameter> 6.6.1.0.0 </parameter>
  <parameter> 6.6.4.0.0 </parameter>
  <parameter> 6.7.4.0.0 </parameter>
</instruction>
<instruction>
  <name> 6.7.0.0.0 </name>
  <action> compute cup size </action>
  <parameter> 6.6.1.0.0 </parameter>
  <parameter> 6.6.2.0.0 </parameter>
  <parameter> 6.7.1.0.0 </parameter>
</instruction>
```

Fig. 13. Extract of instruction “compute size” refinement.
7. Conclusion and future works

We have presented in this article a framework for a web-based CAD system. We have emphasized on the distributed architecture by proposing a replicated architecture based on a multi-level language. We have continued by presenting the multi-level framework composed by several modules. We have explained the role of each of them and we have emphasized on the communication between these modules. We have then presented the concepts and the usefulness of such a language and proposed a naming mechanism to allow the client and the server to work on the same model. This architecture has been used to implement two applications. They deal with art of entertainment design and foundry. We actually work to extend these applications in a more industrial context. In future works, we will try to validate this framework to collaborative works.

Acknowledgements

DIJA is a project with grants from European Community, the Ardennes department and the Champagne-Ardennes region.

References

[18] Cano P, Torres JC, Velasco F. Progressive transmission of polyhedral solids using a hierarchical representation scheme. in: Proceedings of the Eleventh International Conference in Central Europe on...


Laurent Denis is a PhD student in the CMCAO team of IFTS, Reims University, France. He received his master of computer science from Metz-Nancy University in 1999. His research interests include computer aided functional design and geometric modelling.

Yvon Gardan, director of IFTS and head of CMCAO team, has been working on CAD/CAM for more than 20 years. His main domains of interest are: modelling (geometrical and functional), man-machine interface for CAD/CAM, design intent modelling. He is the author or co-author of about 10 books and the Chairman of MICAD conferences. He is an expert for the French Government in the areas of innovation using CAD. He has leaded many industrial or European projects.

Estelle Perrin is an associate professor in the CMCAO team of Metz University, France. She received her PhD degree in computer science from Metz University in 1995. Her research interests include computer aided functional design, Boolean operations and geometric modelling.