A SOFTWARE ARCHITECTURE FOR COLLABORATIVE VIRTUAL PROTOTYPING

Patrice TORGUET [†] , Olivier BALET [†/‡] , René CAUBET [†]

[†] IRIT, Paul Sabatier University, 118, Route de Narbonne
31062 Toulouse Cedex, FRANCE
torguet@irit.fr
http://www.irit.fr/~Patrice.Torguet

[‡] CISI, 13, Rue Villet, Z.I. du Palays
31029 Toulouse, FRANCE
Olivier.Balet@cisi.cnes.fr
http://www.ensica.fr/~balet

ABSTRACT

Prototype design and testing is an indispensable stage of any project development in many fields of activity, such as aeronautical, spatial, automotive industries or architecture. Scientists and engineers rely on prototyping for a visual confirmation and validation of both their ideas and concepts. This paper describes the collaborative and distributed aspects of PROVIS, a system for prototyping virtual systems. The main goal of our work is to allow designers to replace physical mock-ups by virtual ones in order to test the integration and space requirements of the model components. To this end, PROVIS combines a "desktop VR" interface with an interactive 3D display. While discussing both PROVIS software and hardware distributed architectures, the paper describes how to implement a complete computer system which allows users to collaboratively build and interact with virtual models while testing maintenance procedures and component accessibility.

Key Words: Distributed Virtual Reality, Interactive Techniques, Virtual Prototyping, Collaborative Working

INTRODUCTION

Nowadays, physical mock-ups are used in a wide scope of activities (aerospace, automotive manufacturing or architecture). They still remain the only solution for testing integration, space requirements and accessibility of their equipment Moreover, they represent long, fastidious and, sometimes, expensive tasks. Finally, physical mock-ups, once realized, are neither flexible nor easy to manipulate.

Considering these observations, CNES[1], CISI[2] and IRIT have jointly launched the PROVIS (PROtotyping VIrtual Systems) research project in 1995. The main aim of this project was to find solutions which would allow satellite designers to create, manipulate and study their models by using digital mock-ups called prototypes.

In order to increase the efficiency of this experiment and to offer an interface as user-friendly as possible, we had to use leading-edge technological concepts, in particular a "desktop VR" interface associated with high-level commands. Actually, we mainly wanted to avoid users to learn a new system for prototyping tasks.

Most of the systems which manage 3D environments are controlled by 2D GUIs. The

[1] The French space agency

[2] The computer science division of the French Atomic Energy Authority Industry Group (CEA)

lack of correlation between manipulation and effect and the great distance between the mental image users have and edited models are the major drawbacks of this solution. On the other side, full 3D interfaces are more often unsuited for industrial applications. For example, the menu system is sometimes replaced by 3D menus [Butterworth et al. 92] floating in the virtual environment without providing more power than its 2D equivalent. Moreover, it takes much longer to activate a 3D button through a 3D sensor than to activate its 2D equivalent with a conventional mouse. These are the reasons why we chose to create a fully open software architecture which will allow the creation of mixed interfaces based on the concept of autonomous entities [Balet et al. 96].

Moreover, we wanted to allow several teams, based in different distant places, to study and construct virtual prototypes in a collaborative way in order to support and enhance concurrent engineering processes. However, developing collaborative virtual applications is a complex time consuming task. In order to develop such applications, the programmer has to be proficient in network, graphics, device handlers and user interface programming. Moreover, network based programs are inherently more difficult to program and debug than sequential ones. In order to simplify this task, PROVIS is based on VIPER (VIrtuality Programming EnviRonment), a system which enables the rapid and easy development of distributed virtual environments.

In previous papers we have detailed PROVIS genetic algorithm base interaction schemes as well as its GUI [Balet et al. 97] and VIPER architecture [Torguet et al. 96, Balet et al. 96]. In this paper, we detail PROVIS distributed and collaborative aspects as well as its relations with our distributed VR system: VIPER.

PROVIS ARCHITECTURE

PROVIS has been designed to allow non-expert users to build virtual prototypes which can be used to test the integration and space requirements of equipment, for instance, or to define and operate kinematics between the mobile parts of a system, while exploring and expanding the on-line documentation.
In addition to an intuitive use which derives from strong interactivity, the main worth of this package is that it allows a qualitative assessment of the prototype components (integration, space requirements, weights, electric consumption, etc.). The 3D graphical representation is actually the visible part of a component featuring other attributes: physical attributes (weight, etc.), functional attributes (object belonging to a subsystem, etc.), documentary attributes (URL links, text and audio comments, etc.). The graphical envelope is an access metaphor to other data which are accessed by physically designating such or such component.

Furthermore, with PROVIS, several designers located on remote sites can work on the same shared prototype and confront their conceptions. This is the reason why PROVIS is based on the VIPER architecture [Torguet et al. 96] in order to deal with cooperative working and data distribution.

VIPER Architecture

VIPER is a generic Distributed Virtual Reality (DVR) system, like AVIARY [Snowdon-West94], DIVE [Hagsand96] or VLNET [Pandzic et al. 95], which has been created in order to design any kind of DVR applications.

The virtual environment model

VIPER uses two main components in order to define a virtual environment: *entities* which encapsulate graphical objects and their behaviors; and *stimuli* which is the base of interaction and information exchange between entities. Both of those components are brought together in a *virtual universe* (Fig. 1).

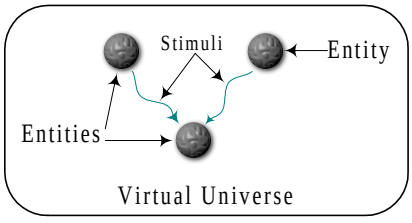

Figure 1: Virtual environment model

Moreover, in order to keep the model consistent, *avatar* entities have been introduced to act as interfaces between the virtual environment and users or applications.

The purpose of this structure is to simplify the definition of distribution patterns. Autonomous entities lead to a perfect encapsulation of both the behavior and the state of an entity, and therefore facilitate their distribution: such an entity can execute its behavior on any site communicating with other entities through well defined stimuli.

The units of communication, *stimuli* (phenomena or events perceptible by an entity), are exchanged through media, called *stimuli spaces*. The stimuli spaces have been introduced in order to allow communications and interactions between many entities simultaneously. Each stimuli space is in fact a projection of the environment along a specific type of stimulus (3D shape space, sound space...). An entity receives perceptible stimuli (visible shapes, near sounds...) through *sensors* and acts on its environment through *effectors* (producing new stimuli) (Fig. 2).

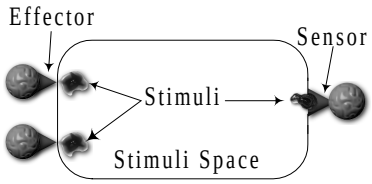

Figure 2: Communication model

The entity model

We have introduced the entity paradigm in order to define in a generic way every kind of scene components. Entities can represent virtual objects, avatars, applications or 3D tools.

An entity is defined by its *behavior*, a set of *attributes*, a set of *sensors* and a set of *effectors*. Both sensors and effectors allow an entity to communicate with its surrounding environment while its attributes define its internal state. This model borrows a lot from behavioral simulation architectures [Wilhelms-Skinner90] where very complex behaviors were successfully implemented (e.g. animal flocks, ant farms...). This is the reason why we chose such a model for the design of complex and dynamic virtual environments.

The behavioral model

Each entity owns a more or less complex behavior (which can be dynamically changed) built with connected behavior components and modeled by an object oriented Petri net [Genrich-Lautenbach81]. Those components can be triggered by a sensor or other components through internal communications. They are either *system components* (hardware dependent) or *application components* (hardware independent) (Fig. 3).

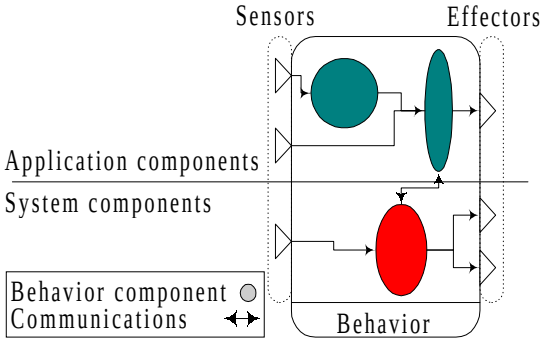

Figure 3: The behavioral model of an entity

For instance, the behavior of a buzzer can be modeled with a simple application component, which generates a sine signal, and a system component which sends an incoming PCM signal to the sound card of the workstation it's running on. The system component can be processed by another computer if the one which runs the application component has no audio capabilities.

VIPER software architecture

In order to better suit the structure of our virtual environments, we have adopted an object-oriented design for our system. Classical advantages of object oriented languages (encapsulation, reuse...) and most of all,

inheritance are very interesting to model entity and stimuli classes. The C++ programming language has been chosen because of its availability on most hardware platforms (from low-end PCs to multiprocessors).

In order to simplify the developer task we have decided to offer the programmer a set of generic classes of concurrent aggregates [Moisan et al. 93]. Those aggregates encapsulate object distribution mechanisms for their elements and remote access over a network. A number of distribution patterns used in distributed VR have been implemented into aggregate classes: active replication [Hagsand96], replication on demand [Singh et al. 94], "mirror worlds" like distribution patterns [Macedonia et al. 95], etc. Virtual universes and stimuli spaces have been defined respectively as concurrent aggregates of entities and stimuli.

The software architecture of VIPER consists in five layers (Fig. 4).

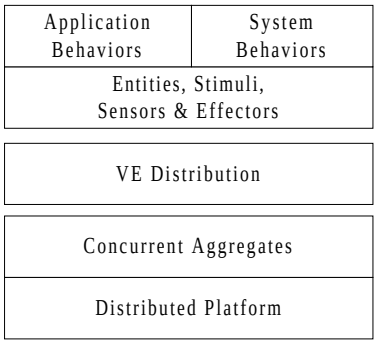

Figure 4 : The software architecture of VIPER.

The two first layers are the kernel of VIPER:

The bottom most layer, the distributed platform, is composed of a set of networked sites (workstations, multicomputers...). This layer encapsulates the communication system and each site description (sound capabilities, graphics hardware, operating system information, etc.). VIPER requires that the communication system provides reliable (even though unreliable communication can be used in some parts of the system) point to point (unicasting) and group (multicasting) communications.

Above this layer, an object-oriented concurrent programming environment encapsulates data distribution using concurrent aggregates.
The three last layers are the API (Application Programming Interface) of VIPER and define two programming levels:

The third layer allows the definition of specific virtual environment distribution patterns. Distribution and remote access mechanisms can be chosen or redefined from a library of existing classes of concurrent objects (based on the second layer). Thus, the developer defines new stimuli spaces and new virtual universes. And therefore, he/she is able to optimize thoroughly his/her application.

Using both topmost layers, a developer can describe a virtual environment (entities, their interactions and their behaviors) as if his/her application was a sequential object oriented program. She or he defines new classes of entities that inherit from existing ones. In fact, this programming level totally hides the distributed aspects of the application.

We will now describe the distribution of a virtual environment using VIPER. Showing how distributed virtual universes and distributed stimuli spaces can solve many problems of such distributed environments. We will present some of the distribution patterns offered by VIPER along this description.

Distributing entities

When distributing a virtual environment, obviously the first data to distribute is entities. VIPER distributes entities using distributed virtual universes (DVU). Each DVU is a concurrent aggregate of entities. It defines a naming function for entities (an entity Id is composed of its DVU Id followed by its creation site Id and a sequential number), a distributing function which tells where (on which site) each entity is stored and possibly some remote access functions for entities.

We have currently defined two types of DVU.

The passive distributed virtual universe holds

entities that can't move from site to site (they are bound to a specific site). New entities can be added dynamically on specific sites and new sites can join the DVU. Avatars obviously belong to this type of universe. Such a DVU only provides a simple distribution function which extracts the site Id from an entity Id.

The active distributed virtual universe where entities can migrate from a site to another one. An example of such entities can be a tennis ball which will be simulated on the site of player A when it is near the player (in the virtual world) and on the site of player B when it is near him/her. Such a DVU provides a distribution function (implemented by a replicated synchronized table which contains for each entity of the DVU: an entity Id and the Id of the site which currently owns the entity) and remote access methods used to send and receive entities.

Many DVU can be combined in order to create a complex virtual universe with site-bound and migrating entities (Fig. 5).

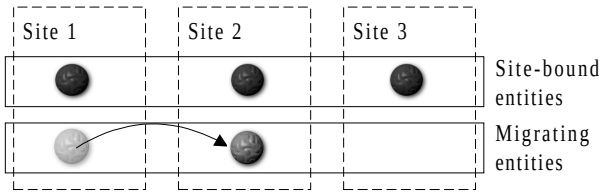

Figure 5: Some distributed virtual universes

Entities communication management

As soon as two or more entities enter the environment, a first problem arises. Each entity must be aware of the others in order to interact with them. VIPER solves this problem with a distributed stimuli space, called the *shape space*, which uses an active replication distribution pattern.

Each entity owns a *shape* which is composed of one or many geometric 3D shapes (which are used as different *level of details* or as multiple views of a same entity — e.g.: a data set which can either be seen as a bar graph or a pie chart), a position, an orientation and a scaling factor in relation to a father object coordinate system (the shape space is a hierarchy of shapes). Each time an entity modifies its shape (movement, change of parent or deformation) the stimuli space replicates those changes on any site using a shape space dedicated communication group (Fig. 6). Then each entity which owns a shape sensor can use this hierarchy in order to present a view of the world to its user (avatar) or to detect other objects (e.g. to detect collisions).

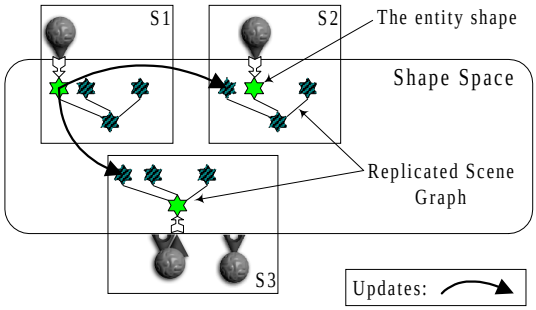

Figure 6: A shape space

It is important to note that all communications are hidden in the stimuli space and therefore an entity, as well as the programmer of the entity class, isn't concerned by communication management.

Moreover, other type of group communications (sound, video, etc.) can easily be added to a virtual environment if other type of stimuli spaces are defined (sound space, video space, etc.). Indeed we have currently defined both sound and video spaces. These stimuli spaces work like the shape space: for instance, each entity can own a sound icon which is replicated on all sound capable sites while sound changes are synchronized.

Direct interaction (i.e. when an entity manipulates other entities) in a distributed context is another problem: interacting entities may happen to reside on different sites. We have decided to model those interactions by exchanging *orders* (a subclass of stimuli) between entities within the framework of a specific distributed stimuli space. This space, called the *order space*, only sends orders to interested entity sites using point to point

communications. Indeed, as each entity Id can be translated to a site Id where the entity is stored (thanks to DVU distribution functions), we can send an order to the correct site (Fig. 7).

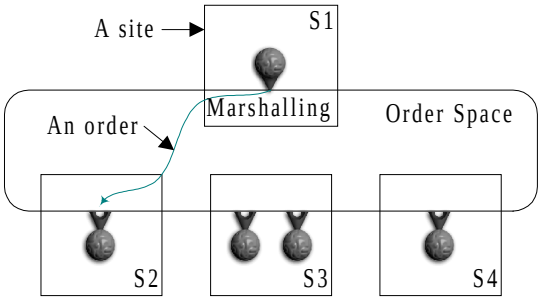

Figure 7: A distributed order space

All the previously defined distribution mechanisms are available through generic classes of distributed sets (concurrent aggregates) which can be used through generic instantiation (declaratively creating stimuli spaces or DVU) or through inheritance (implementing application specific optimizations – e.g. for a vehicle simulation we can implement dead reckoning) in order to create specific virtual environments.

For PROVIS, we use a passive DVU (for designers avatars and PROVIS specific entities), an active DVU (for prototype parts), a shape space (for the visual communication), a sound space, a video space and an order space.

PROVIS Entities

In its basic version, PROVIS is defined by a set of 3 main entities:

- The Builder: The aim of this entity is to provide the user with high-level commands in order to build 3D prototypes. It manages the model library, the interaction process and the application specific functions (components assembly, documentation, etc.). In fact, it specifies the application functionality.

Fig. 8 presents the simplified model of this entity: Several behavioral components (input modules) analyze and control incoming stimuli such as gestures or sound in order to convert them into normalized events. These events are sent to the *Event Manager* which merges them into a single command which is then sent to the modeler. Thanks to its multimodal event manager, PROVIS allows the user to combine several input devices (dataglove, speech recognition system, spaceball, flying mouse, mouse...) in order to build a single command. For instance, he or she can say to the system : "Put this on the table", where *this* stands for a bottle (previously selected with his or her flying mouse) which is retrieved in the dialog history thanks to its syntactic and semantic attributes [Bolt80].

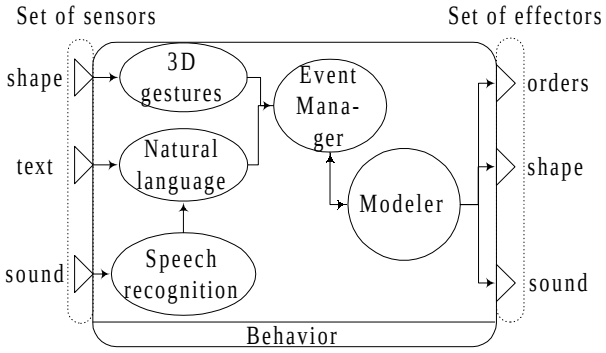

Figure 8: A simplified view of the Builder's behavioral components

Moreover, the multimodal event manager allows information coming from different users to be combined into a single command (Fig. 9). We use a homogeneous command description to define the signature of every command of our system. For instance, the MOVE command is specified as follows:

```
MOVE (Object, Force, ...)
[
        MULTI-USER;
        DELAY = 100;
]
```

Thus, MOVE can receive several forces as parameters. This command is a MULTI-USER one, i.e. it can receives its parameters from different users (This is very useful for maintenance tests when an element requires several users to be moved). The delay attribute specifies the delay (in milliseconds) between

the receipt and the execution of the command. This attribute is used by the event manager to define how long it will wait for incoming events before sending the complete command (Fig. 9). Please note that the multimodal event manager component is a default behavioral component of every prototype element. This allows these elements to be manipulated by several users or by a single user using both his hands at the same time.

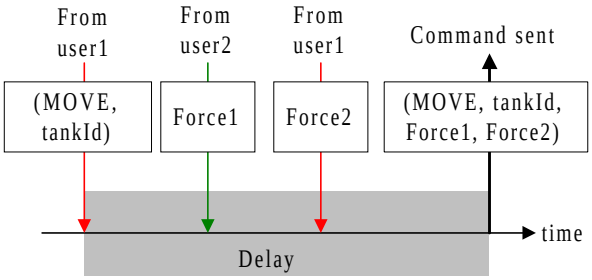

Figure 9: The event manager multimodal resolution

The modeler component contains procedures that affect the scene such as geometrical procedures or declarative functions. The aim of the declarative functions [Gaildrat et al. 93] is to offer the user the ability to describe scenes without giving numerical data. It enables the user to express his mental image of the scene in a quasi-natural language, including high level commands, deictics and fuzzy parameters : "Put the bottle on the table", "Move it to the left".

After studies in ergonomics, we have decided to limit our gestural language to a set of 3 kind of commands : select, rotate and translate. In order to clarify the 3D display, we have also decided to assign a simple 3D shape (a colored 3D pointer) to the avatars which represent other users. Moreover, this kind of avatar can also be represented by a 2D window (which displays the user's face video signal) and the user's digitized voice (Fig. 10).

- The Expert: Its aim is to check the coherence of a prototype: weight assessment, power consumption and construction integrity. This entity is supposed to contain the experts' knowledge. It uses several attributes (semantic attributes, etc.) of the prototype components in order to assess the validity of the prototype.

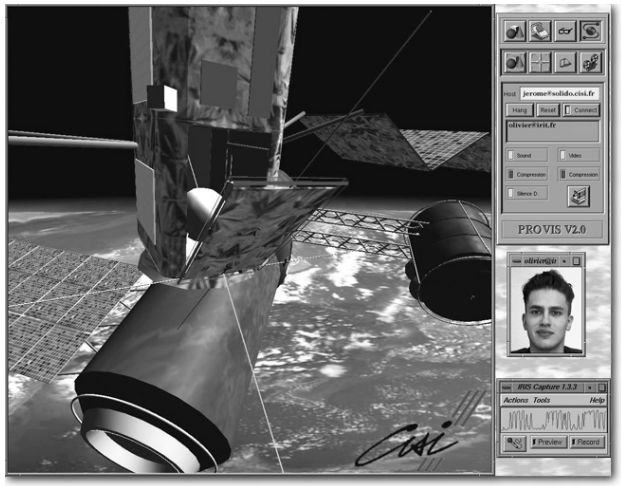

Figure 10: Building a satellite with a remote user

- The 2D GUI : The graphical user interface is in fact a simple frame where other entities can add or remove their own widgets (Fig. 11.a). For instance, an entity adds its widgets by sending orders to a 2D GUI when its 3D shape is selected (Fig. 11.b). Thereafter, an entity will receive orders through its order sensor, each time its widgets are activated. The 2D GUI has been designed in order to work either on a workstation running X11 or on a PC running Windows'95 or Windows NT.

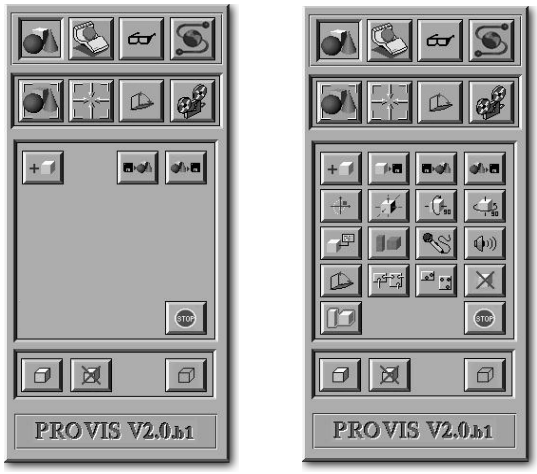

Figure 11: (a) GUI with the Builder's widgets - (b) GUI with the selected entity's widgets

Thanks to the distributed architecture, the GUI entity can run on a workstation while the

designer is working on another one. This is very useful when, for instance, the workstation used by the designer is displaying a full screen stereo view of the scene. Indeed, some workstations can't display at the same time, stereoscopic views and 2D widgets on the same screen for hardware reasons.

Hardware Architecture

Interacting with three dimensional worlds [Wloka95] does not fundamentally differ from interacting with two dimensional worlds. In both environments, the user has to select objects, enter values, set positions, orientations or scales. However, using common devices (such as a mouse) for 3D manipulations more often slows down the user's work. That's why we tested both a spaceball and a dataglove to interact intuitively with the 3D entities. Moreover, we decided to provide the user with a stereoscopic vision of the virtual environment (through stereo glasses or HMDs) in order to take advantage of the third dimension's added power. During the PROVIS evaluation phase, users expressed their preference for the spaceball and the stereo glasses. Therefore, we finally decided to use a "desktop VR" interface (Fig. 12) which allows the operator to work during several hours without the incomfort which results from datagloves (muscular tiredness) and immersive head-mounted displays (eye strain and excessive weight). Stereoscopic display is achieved thanks to stereo glasses with a built-in 3D sensor that allows viewpoint movements to be linked to the operator's head.

Figure 12: Chosen interaction devices

Moreover, a microphone feeds speech to a PC with both hardware and software for speech recognition. The recognized sentences are in turn sent to the host computer.

Evaluations were done on both Maximum Impact and Onyx/RE^2 SGI workstations connected to a PC computer for speech recognition.

WORKING WITH PROVIS

Owing to the previously described architectures, PROVIS allows users to manipulate 3D objects in a natural way. Also, PROVIS features several 3D widgets [Conner et al. 92] which speed up the prototyping task. For instance, we provide a 3D cyclic selector (Fig. 13) which allows users to interactively link elements by selecting the 3D normalized representation of the joint type they choose.

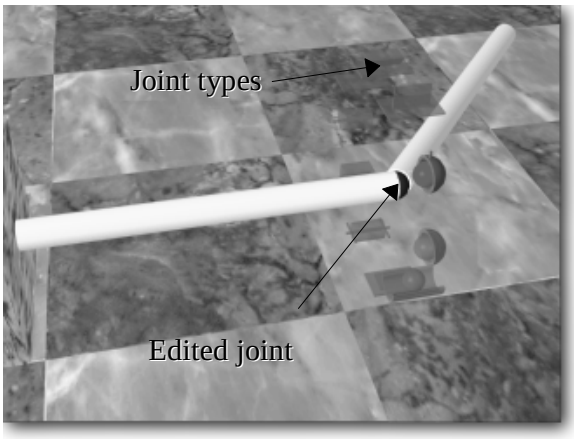

Figure 13: The joint type selector

However, we had to introduce more complex tools to manipulate complex objects such as wires or flexible pipes. Routing these kind of objects was one of the main problems which had to be solved before really using virtual prototyping for an industrial project. In our system, these objects are modeled as articulated chains defined by tubes, joints and constraints (flexibility, etc.). Unlike the motion of *passive* systems like falling objects or elementary components, the motion of articulated objects, human or robot arm for the purpose of manipulation are *motions with intentions* [Koga et al. 94].

We had to offer designers two kinds of interaction concepts :
- Direct manipulation: The user uses a 3D pointer (Fig. 14) to interact with an articulated object, grasping either the end or any other part of the object.

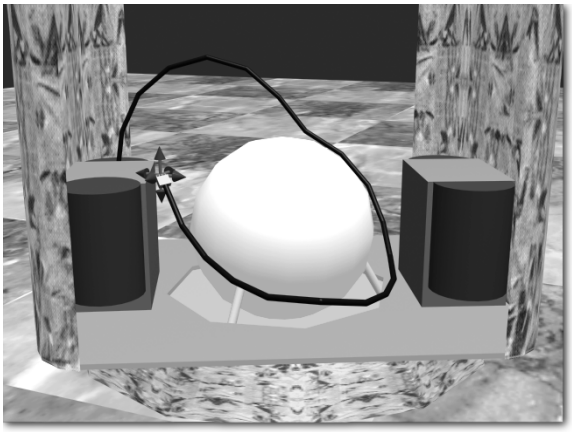

Figure 14: Direct manipulation of a quite rigid wire (20 joints) within the satellite platform

- Task-level manipulation: The user chooses a wire, defines a source and a target, the favorite zones, the zones not allowed and launches the task resolution. Then, the wire plans a collision-free path which respects the specified constrains. Fig. 15a presents the different parameters which have been interactively defined by the user. The sphere represents a zone which must be avoided while planning the wire path. The clip is a zone in which the wire has to pass through. Note that the clip used in the following figure (Fig. 15.a) is just a position constraint and that we also provide users with another kind of clip which represents both position and rotation constraints. Even though the solution presented (Fig. 15.b) has implied the clip rotation, it respects the position constraint.

Several methods have already been proposed to manage articulated figures [Badler et al. 91]. Based on inverse kinematics [Zhao-Badler94] or inverse dynamics [Isaacs-Cohen87] models, for example, these methods never offer an intuitive way to define internal and global constraints. We have proposed [Balet et al. 97] an original use of genetic algorithms in order to define behaviors for articulated objects so that we can manage in a very homogeneous way, interactions with fully constrained articulated objects.

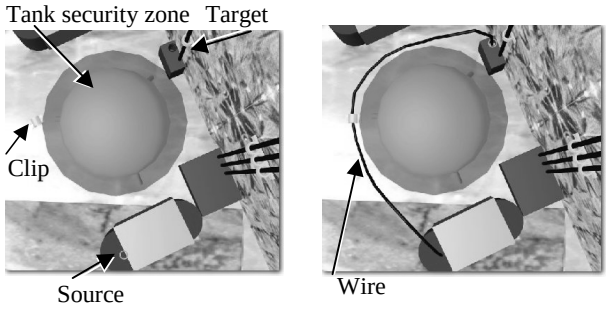

Figure 15: (a) Defining the task parameters, (b) The computed solution

CONCLUSION AND FUTURE WORK

In this paper, we have presented PROVIS distributed architecture and collaborative aspects. This interactive 3D software solution can instill collaborative virtual prototyping in the various phases of every project, thereby easing and speeding up the designers' work.

We have first described the distributed virtual environment model we have implemented on an heterogeneous workstation network. This model differs from other proposed models in the literature in at least two ways. It emphasizes on a separation between the behavioral model of a virtual environment and its interaction model in order to improve virtual environment modularity. Moreover VIPER, its companion software architecture, allows distribution patterns to be chosen and/or redefined by the virtual environment developer.

We have then described how the different entities defined for PROVIS (the builder, the expert and the 2D GUI) allows collaborative work between several engineer teams.

Then, we described how interacting with complex (i.e. articulated) virtual prototype components can be intuitively done thanks to PROVIS interaction features.

We are currently evaluating the ergonomic of several more 3D widgets in order to provide

CNES with a definitive solution which will be used soon for the design of next generation satellites.

REFERENCES

Balet O., Torguet P., Gaildrat V., Caubet R. 1996 Autonomous entities in distributed virtual environments, *Multimedia Modeling – Towards The Information Superhighway, Proceedings of MMM'96*, World Scientific, pp 457-471.

Balet O., Luga H., Duthen Y., Caubet R. 1997 PROVIS: A platform for virtual prototyping and maintenance tests, *Proceedings of Computer Animation'97*, Geneva, Switzerland, June, pp 39-47.

Badler N.I., Basky B.A., Zeltzer D. 1991 *Make them move: Mechanics, control, and animation of articulated figure*, Morgan Kaufmann.

Bolt R.A. 1980 Put-that-there: voice and gesture at the graphics interface, *Computer Graphics*, Vol. 14, no 3, ACM Press, July, pp 262-270.

Butterworth J., Davidson A., Hench S. Olano T.M. 1992 3DM: A three dimensional modeler using a head-mounted display, *Proceedings of the 1992 Symposium on Interactive 3D Graphics*, ACM Press, March, pp 135-138 .

Conner D.B., Snibbe S.S., Herndon K.P., Robbins D.C., Zeleznik R.C., Van Dam A. 1992 Three-dimensional widgets, *Proceedings of the 1992 Symposium on Interactive 3D Graphics*, ACM Press, March, pp 183-188.

Gaildrat V., Caubet R., Rubio F. 1993 Declarative scene modelling with dynamic links and decision rules distributed among the objects, *Proceedings of ICCG 93*, IFIP, Bombay, India, February, pp 165-178.

Genrich H.J., Lautenbach K. 1981 System modelling with high-level Petri nets, *Theoretical Computer Science*, Vol. 13, pp 109-136.

Hagsand O. 1996 Interactive Multiuser Ves in the DIVE System, *IEEE Multimedia*, Spring 96 issue, IEEE Computer Society Press, pp 30-39.

Isaacs P.M., Cohen M.F., 1987 Controlling dynamic simulation with kinematic constraints, behaviour functions and inverse dynamics, *Proceedings of SIGGRAPH'87*, ACM Press, pp 215-224.

Koga Y., Kondo K., Kuffner J., Latombe J.C. 1994 Planning motions with intentions, *Proceedings of SIGGRAPH'94*, ACM Press, pp 395-408.

Macedonia M.R., Zyda M.J., Pratt D.R., Brutzman D.P., Barham P.T. 1995 Exploiting Reality with Multicast Groups: A Network Architecture for Large-scale Virtual Environments, *IEEE Computer Graphics & Applications*, IEEE Computer Society Press, September, pp 38-45.

Moisan B., Duthen Y., Caubet R. 1993 Tools for SPMD object-oriented programming, *Proceedings of EUROMICRO'93*, North-Holland, pp 189-196.

Pandzic I.S., Capin T.K., Magnenat-Thalmann N., Thalmann D. 1995 VLNET: A Networked Multimedia 3D Environment with Virtual Humans, *Proceedings of MMM'95*, World Scientific, pp 21-32.

Singh G., Serra L., Png W., Ng H. 1994 BrickNet: A Software Toolkit for Network-Based Virtual Worlds, *Presence*, Vol. 3, no 1, MIT Press, pp 19-34.

Snowdon D.N., West A.J. 1994 AVIARY: Design issues for Future Large-Scale Virtual Environments, *Presence*, Vol. 3, no 4, MIT Press, pp 288-308.

Torguet P., Rubio F., Gaildrat V., Caubet R. 1996 Multi-user interactions in the context of concurrent virtual world modeling, *Virtual Environments and Scientific Visualization '96*, Springer Computer Science, pp 121-136.

Wilhelms J., Skinner R. 1990 A notion for interactive behavioural animation control, *Computer Graphics and Applications*, IEEE Computer Society Press, Vol. 10, no 3, May, pp 14-22.

Wloka M. 1995 Interacting with Virtual Reality, *In Virtual Prototyping – Virtual Environments and the Product Development Process*, Chapman & Hall.

Zhao J., Badler N.I. 1994 Inverse kinematics positioning using non-linear programming for highly articulated structure, *ACM Transactions on Graphics*, Vol. 13, no 4, ACM Press, October.