VIPER (VIrtuality Programming EnviRonment): A virtual reality applications design platform

Patrice TORGUET, René CAUBET

I.R.I.T. Université Paul Sabatier,
118, Route de Narbonne, 31062 TOULOUSE CEDEX, FRANCE
e-mail: torguet@irit.fr

Abstract: In this paper we present a new virtual reality applications design platform, VIPER, which emphasizes distributed aspects of virtual environments. Our goal is to propose the developer a generic system which can be specialized to suit an application, while minimizing his work all along the virtual environment definition. We describe the general structure of an environment in VIPER, and discuss the distribution of such an environment.

Keywords: virtual environments, application design platform, distribution, programming environment.

1. Introduction

In order to increase the realism of virtual environments, one solution is to allow many users as well as simulated entities to interact in a shared virtual environment. Such a virtual environment appears more real from each user viewpoint thanks to the rest of the environment liveliness. Applications and worlds in these environments are difficult to design and simulate mainly because of their distributed properties.

Our goal is to define a generic system for virtual reality application designing and virtual environment simulation. This system, called VIPER, has to simulate multiparticipant virtual worlds in real-time, thanks to a distributed platform.

In this paper we present the general structure of a virtual environment in VIPER as well as the framework of the system software architecture.

2. Survey of existing similar systems

Virtual environments operating systems can be roughly divided into two classes.

The first class is composed of systems dedicated to a certain type of application. Good examples of this class are: NPSNET [Zyda93] which is dedicated to military simulations and IPA's Transputer-based virtual reality workstation [Strommer93] which is dedicated to industrial robot control and programming. Those systems have developed domain specific accelerating techniques: specific hardware (dedicated Transputer architecture of IPA's VR workstation) and optimization algorithms (dead reckoning for NPSNET like systems). Those systems are very efficient for their specific domain but seem difficult to reuse for other applications.

Application generic systems compose the second class. Good examples of this class are: DIVE [Carlsson93], VEOS [Bricken93] and AVIARY [West92]. Those systems have to be the most generic possible in order to be easily used to design any application. This genericity, however, trades off with the general efficiency of the final application. For example in most of these systems, optimization algorithms are difficult to implement.

In order to better deal with efficiency within a generic system, we think that tools should be proposed to the virtual environment developer. Systems like WAVES [Kazman93a], have tried to implement optimizations in the core of the system. We rather think that the definition of such optimizations should be available for the virtual environment developer, either declaratively (the developer chooses an available optimization for his current environment) or explicitly (the developer implements a new acceleration, eventually based on an existent one).

3. *Virtual environment structure*

VIPER is aimed for the design of every application that is based on a virtual environment which can be modelled using the following general structure.

In VIPER, a virtual environment is made of entities, stimuli and a virtual universe (Fig. 3.1).

- The *"entity"* paradigm allows uniform management of virtual worlds scenery, virtual objects and *"clones"*. Entities are autonomous and own a set of attributes and behaviours. They are conceptually grouped in *"families"* (a set of entities which own the same attributes and behaviours). Our system is best suited for homogeneous virtual environments (few families made of many instances).
- The *"stimuli"* convey interactions between entities.
- The *"virtual universe"* is the three dimensional environment where entities interact and behave.

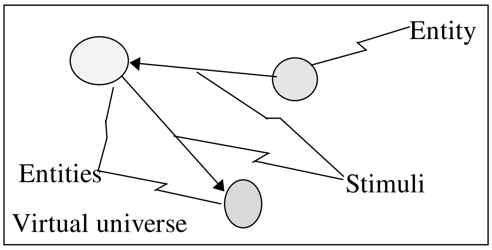

Figure 3.1: Virtual environment structure

A *"clone"* is a special type of entity which behaves as an interface between a user [Mouli93], an application [West92] [Carlsson93] or a robot and the virtual universe. An example of an application clone can be a modeller such as DeM²ons [Gaildrat93] a declarative multimodal modeller which is to be integrated with VIPER. The modeller entity will be able to provide some services to a user (in fact to a user's clone) in order to build a virtual world.

The purpose of this structure is to ease the definition of distribution schemes. Autonomous entities leads to a perfect encapsulation of the behaviour and state of an entity, and therefore ease distribution of entities: such an entity can execute its behaviour on a specific site, communicating with other entities through well defined stimuli.

In this part, in order to describe the structure of entities, we will present the inter-entity communication mechanism and the behavioural part of entities.

Entities and their environment interact in two main ways:

- from the environment to entities: a change in the environment produces a stimulus which is perceived by some entities. A stimulus is received by a specific sensor which interprets and then transmits information to the behavioural part of an entity.

- from each entity to the environment: once the information has been analysed, the behavioural part is able to change the internal state of the entity and if necessary commands actions to the effectors of the entity. Effectors act on the environment and while modifying it create new stimuli.

We propose to model these interactions with four elements: sensors, effectors, images and image spaces (Fig. 3.2). This model deals with environment internal interactions as well as those happening with the real world (e. g. for *"clone"* - user communication, devices such as datagloves and tracking systems are encapsulated in sensors).

To each type of stimuli, we associate a quadruplet: (image, image space, sensor, effector). An image is the perceptible part of an entity in the environment, in relation to a specific type of stimuli (e.g. a 3D shape is the perceptible part of an entity in relation to visual stimuli). An image space is mainly the set of images related to a certain type of stimuli. A sensor gets a "snapshot" of an image space which it filtrates and interprets. While executing an action an effector produces an image and then puts it into an image space. In fact, to each effector is first associated a unique image in the image space which is modified afterwards.

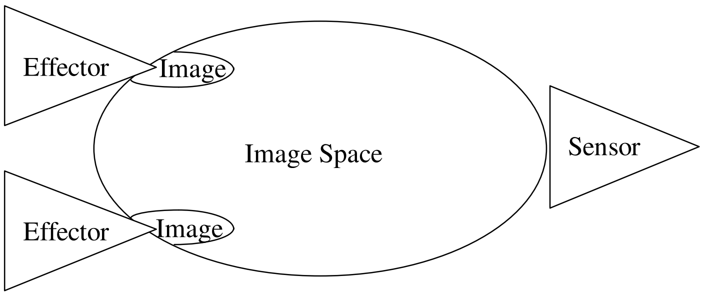

Figure 3.2: Inter-entity communication mechanism

Introducing an image space, which acts as a mediator in the inter-entity communication, is very useful in order to distribute a virtual environment (as will be explained in section 4.3).

The behavioural part of an entity consists in two components (Fig. 3.3): a *"reflex"* component, where entity reflexes to stimuli are defined, and a *"thinking"* component, where more complex behaviours can be defined. Behaviours can be implemented by finite state machines [Carlsson93], sensors/actuators networks [Wilhelms90] [Panne93], Prolog like rules [Rainjonneau90], intention generators [Xiaoyuan94], classifier systems [Torguet95] as well as interfaces to users, applications or robots.

The reflex component is made of a number of modules. Each module is assigned to a sensor in order to analyse its results and if necessary react, commanding actions to effectors and/or modifying its own state. The thinking component is triggered either by a clock or by any reflex module. This component defines the active side of the behaviour while the reflex component defines the reactive side.

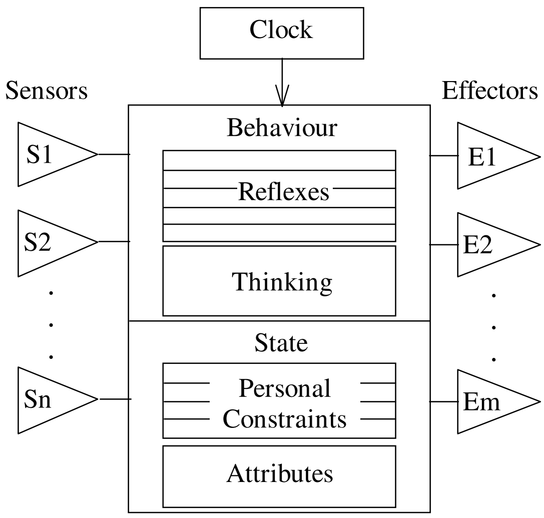

Figure 3.3: An entity

The entity has also its own state: a set of attributes (position, velocity, orientation, mass, 3D shape...) with a set of personal constraints (constraints which act on the entity attributes, e.g.: $velocity \leq maximum\_velocity$).

4. *Software architecture of VIPER*

The software architecture of VIPER consists in four layers (Fig. 4.1):

- At the bottom, the distributed platform, which is composed of a set of sites (workstations, multicomputers...), creates a virtual machine based on a message passing system like PVM [Sunderam90]. This layer encapsulates the communication system used by VIPER.

- Above this layer, an object-oriented concurrent programming environment encapsulates data distribution and an SPMD (single program multiple data) model [Moisan93].

- The third layer is the entity model, already described in the third section of this document.

- The topmost layer allows the definition of specific virtual worlds and the choice of their distribution scheme.

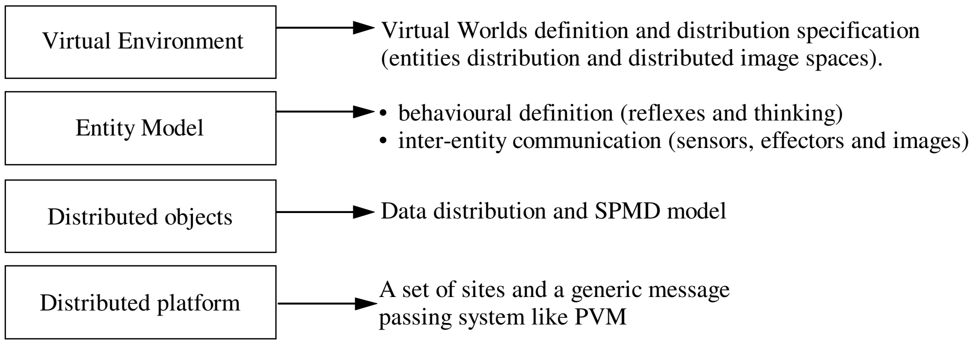

Figure 4.1: The software architecture of VIPER

4.1. The distributed platform

The distributed platform is the set of sites which take part in the simulation of the virtual environment. It is logically divided into four sets of resources: computing, rendering, storage and input/output resources. Each site is at least referenced into one of these sets. For example, a graphics workstation is referenced in all those sets.

Rendering resources are the set of sites which have render specific hardware. A graphics board and a three-dimensional sound rendering board are good examples of rendering resources. These resources are usually connected to sensorial devices such as head mounted displays or headphones. This set is also divided into subsets of similar rendering resources.

Input/output resources are the sites which have input/output ports used by VIPER. This set is divided into three groups: acquisition resources which allow access to virtual reality devices (such as datagloves and tracking systems), standard input/output resources (console) and operating resources which operate external systems (such has robot control boards). Acquisition and operating resources are further divided into subsets of similar devices.

4.2. Distributing an entity over the platform

In order to be easily distributed over the platform, each entity is divided into a number of parts called "sub-entities" (Fig. 4.2). These sub-entities are: rendering, input/output, behavioural and general sub-entities.

Each rendering sub-entity is composed of render specific sensors and attributes. Each input/output sub-entity is composed of acquisition sensors or operating effectors and attributes that are specific to a type of input/output. The storage sub-entity is made of all storage effectors and storage specific attributes. The behavioural sub-entity is composed of all the other sensors and effectors, reflexes, thinking, personal constraints and behavioural specific attributes. The general sub-entity is made of the set of attributes which are not specific to one of the other sub-entities.

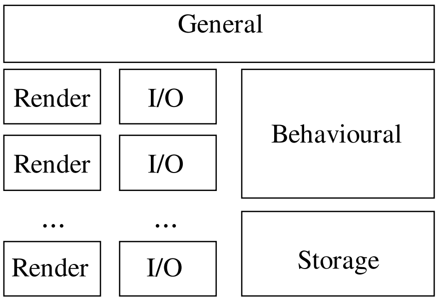

Figure 4.2: An entity divided into sub-entities

Each rendering sub-entity is given to one of the matching rendering resources. Each input/output sub-entity is similarly given to a matching resource. The behavioural sub-entity is given to one of the computing resources. The storage sub-entity is given to one of the storage resources. The general sub-entity is duplicated over all resources where the entity has a sub-entity. Practically, all entity resources ought to refer to the same site (e.g. a multiprocessor graphics workstation) or a few tightly connected sites. And the system always tries to get this property verified.

4.3. Distributing Virtual Worlds

This distribution of an entity introduces a functional parallelism which is internal to the entity. A more general parallelism within the virtual universe can be achieved. A virtual universe is a homogeneous environment inhabited by entities which behave in a similar way.

Object parallelism has been mainly used in existing systems because of its simplicity [Kazman93a] [Snowdon93]. The main advantage of this parallelism is to maintain a good load balancing. Obviously if each site is given the same load of entities each one would become equally loaded. Moreover, a strict encapsulation of entities allows the introduction of dynamic load balancing [Kazman93b].

However, in order to detect and manage direct or indirect interactions (collisions or information exchange in the environment), communication between sites introduces a great number of problems (bottle-necks on an interaction specific site for example).

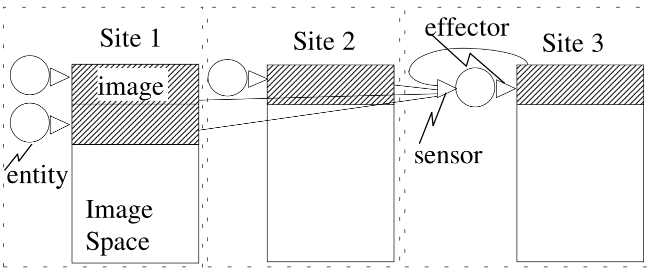

Figure 4.3: A fully distributed image space.

VIPER proposes to solve these problems with distributed image spaces. Communication between entities through image spaces has the advantage of being well defined and easily manageable.

We have currently defined three kind of distributed image spaces:

- fully distributed image spaces. In those image spaces, images are only situated on their entity computing resource and are remotely accessible. For example in Figure 4.3, when each entity effector creates an image of the entity for the first time, each image exist on the site where it has been created. And afterwards, modification of these images are only locally done. The entity on the third site is able to get every images present in the image space. This is done totally transparently: an access to a distant image is exactly the same as an access to a local image as far as the sensor is concerned. The only difference being, obviously, access time. We think that this first type of image space is interesting when there are few sensor doted entities which retrieves information only from time to time, whereas effectors modify images very often. An example of such an image space can be a low frequency radar image space, where entities images are their position in space and sensors are radar which examine the image space with a low frequency.

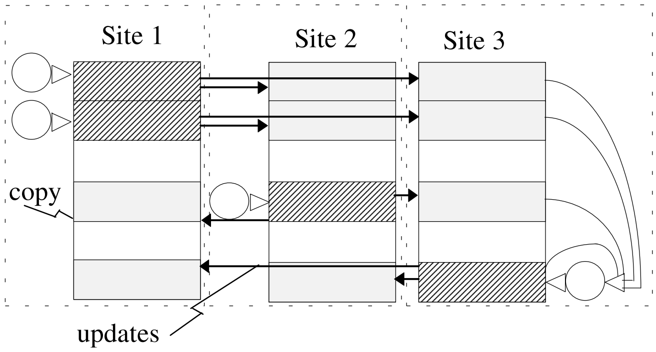

Figure 4.4: A fully duplicated image space

- fully duplicated image spaces, where images are duplicated over all computing resources and image updates are propagated. For example in Figure 4.4, when each effector creates an image, the image is created on every site the image space is accessible from. Each modification of an image is thereafter propagated to all sites via update messages transparently sent by the image space itself. When, for example, the sensor of the third site entity examines the image space, it only looks at local copies of each images. This type of image space is very interesting for image space where there is a lot of sensors sensing the image space often. An example of such an image space is the one used to mediate visual communication between entities. In this image space, entities outputs their changes in appearance (such as a position modification when moving) and other entities sense these changes in order, either to modify their behaviour or to display a view of the 3D world to their users (for user's clones entities).

- smart duplicated image spaces, where images are duplicated over computing resources which needs them (i.e. if there is, currently, a sensor connected to the image space) (Fig. 4.5). This is a derivative of duplicated image spaces, useful when there are few sensors (less than one at each site). An example of such an image space can be a sound mediating image space, which manages distribution of sounds produced by users or virtual objects and listened by users which have sound producing boards.

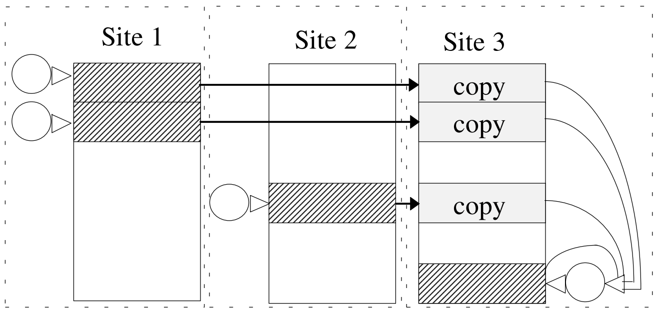

Figure 4.5: A smart duplicated image space.

However, in order to better solve some of the previously mentioned problems we think that we should take into account that, in most communication types, two entities may only interact if they are close enough. This can be done by associating topological information to image spaces.

A logical spatial partition of image spaces seems interesting for worlds which are logically divided into rooms linked by portals (doors, openings...). Each entity is associated to the room in which it is spatially present, and each site will only manage interactions which involve its entities (i.e. needs only the part of each image space which is local to its entities rooms). For example, if we consider a virtual building, an entity may only interact with entities located in the same room or in adjacent rooms (if there are open doors).

A regular space partition can also be considered [Zyda93]. In this case, the virtual world space will be divided into grid squares of a constant size. Such a built grid can either be two dimensional (in case of applications in which altitude doesn't matter much: vehicle simulation for example) or three dimensional (applications where altitude is very important: flight simulation for example). Each entity is associated to the grid zone in which it is spatially present, and each site will only manage interactions which involve its entities (i.e. needs only local parts of each image space). An example of the use of topological information is presented in Figure 4.6. In this figure, the plane is only able to detect two vehicles because the image space, being aware of plane sensors limitations, only take into account a part of the virtual world when listening to vehicle images updates.

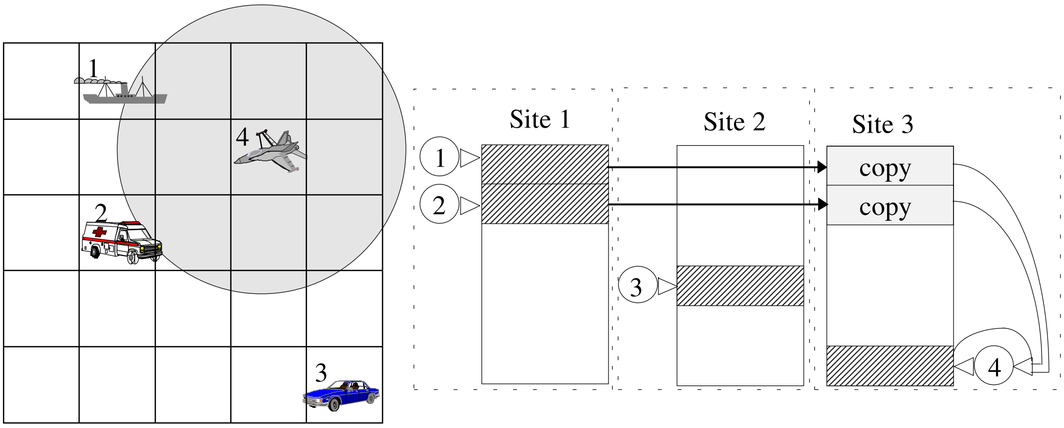

Figure 4.6: A smart duplicated image space with topological information.

An a priori choice for a distribution model is difficult to make. In fact this choice is really application dependent. For example, for a flight simulation a regular image space partition seems interesting, while a logical space partition seems more useful for an architectural walk-through. In VIPER the application developer is able to choose for each world and for each image space a specific distribution scheme. Moreover, the developer can specialize an already defined image space in order to better suit his needs.

Thus, the developer can quickly prototype his application using existing general distributed image spaces. And then, in order to increase efficiency, he may try other distributed image spaces or explicitly implement new optimizations algorithm. For example, the developper of a vehicle simulation application might choose a smart duplicated image space to mediate visual communication between entities (vehicles), then he might prefer a regularly partitionned image space, and eventually implement dead reckoning of vehicles.

5. *First results and Conclusion*

A first implementation of VIPER is being developed. It is based on PVM over an heterogeneous network of workstations (SGIs, HPs and SUNs). The choice of the PVM communication system, has been dictated in order to rapidly prototype VIPER. Indeed, PVM offers interesting tools like xpvm, which ease the definition of parallel applications and their tuning. However, the overhead added by this communication environment doesn't seem to match large scale virtual environments requirements. Thanks to the encapsulation of the communication system, VIPER can, nevertheless, be easily ported over any more efficient communication system.

VIPER is being developed using the C++ programming language which offers very interesting tools in order to develop generic constructs. C++ templates, for example, allows the definition of generic distributed image space which can be parameterized by a specific class of image (and eventually derived), in order to suit application needs.

The entity model, fully distributed image spaces and fully duplicated image spaces have already been implemented and are being benchmarked. Now we are developing our first application: a modeller which gives users the ability to create new shapes while deforming simple ones. This first application will allow us to estimate the efficiency of VIPER when confronted to a real problem.

This first implementation will then be extended with smart image spaces and virtual worlds topological information. The integration of a multicomputer in the platform is also being studied.

References

[Bricken93] W. Bricken, G. Coco, "The VEOS Project", Technical report, Human Interface Technology Lab, University of Washington, 1993.

[Carlsson93] C. Carlsson, O. Hagsand "DIVE - a Multi-User Virtual Reality System", Proceedings of VRAIS'93, Seattle, September 1993.

[Gaildrat93] V. Gaildrat, R. Caubet, F. Rubio "Declarative Scene Modelling with Dynamic Links and Decision Rules Distributed Among the Objects", IFIP International Conference on Computer Graphics ICCG93 Bombay, February 1993.

[Kazman93a] R. Kazman "Making WAVES: On the Design of Architectures for Low-end Distributed Virtual Environments", Proceedings of VRAIS'93, Seattle, September 1993.

[Kazman93b] R. Kazman "Load balancing and latency management in a distributed virtual world", 3rd International conference on Cyberspace, May 1993.

[Moisan93] B. Moisan, Y. Duthen, R. Caubet "Tools for SPMD object-oriented programming" EUROMICRO'93 Barcelona 6-10 September 1993.

[Mouli93] R. Mouli, Y. Duthen, R. Caubet "In VitrAm (In Vitro Animats, a behavioural simulation model)" 2nd IEEE International Workshop RO-MAN'93, Tokyo, 3-5 November 1993.

[Panne93] M. van de Panne, E. Fiume "Sensor-Actuator Networks" Computer Graphics SIGGRAPH'93 proceedings, pp 335-342, 1993.

[Rainjonneau90] S. Rainjonneau, Y. Duthen, R. Caubet "An Object Oriented Approach including an Inference Engine for Behavioural Simulation and Advanced Animation", TOOLS3 Sidney, pp 379-385, December 1990.

[Snowdon93] D. N. Snowdon, A. J. West, T. L. J. Howard "Towards the next generation of human-computer interface", Interface to real and virtual worlds, Montpellier France, pp 399-408, March 1993.

[Strommer93] W. M. Strommer, J. G. Neugebauer, T. Flaig, "Transputer-based virtual reality workstation as implemented for the example of industrial robot control", Interface to real and virtual worlds, Montpellier France, pp 137-146, March 1993.

[Sunderam90] V. Sunderam "PVM: A framework for Parallel Distributed Computing", Concurrency: Practice & Experience Vol. 2 No. 4, December 1990.

[Torguet95] P. Torguet, H. Luga, Y. Duthen, R. Caubet "A model for a user-adaptive response to interactions in Virtual Worlds", submitted to the Graphics Interface '95 conference, 1995.

[West92] A. J. West, T. L. J. Howard, R. J. Hubbold, A. D. Murta, D. N. Snowdon, D. A. Butler "AVIARY - A Generic Virtual Reality Interface for Real Applications", Virtual Reality Systems, sponsored by the British Computer Society, May 1992.

[Wilhelms90] J. Wilhelms, R. Skinner "A *Notion* for Interactive Behavioral Animation Control" IEEE Computer Graphics and Applications, May 1990, pp 14-22.

[Xiaoyuan94] T. Xiaoyuan, D. Terzopoulos "Artificial Fishes: Physics, Locomotion, Perception, Behavior" Computer Graphics SIGGRAPH'94 proceedings, 1994, pp 43-50.

[Zyda93] M. J. Zyda, D. R. Pratt, W. D. Osborne, J. G. Monahan "NPSNET: Real-time Collision Detection and Response" The journal of Visualization and Computer Animation, 4(1), 1993, pp 13-24.