Model-based Specification of Virtual Interaction Environments

A.Celentano, F. Pittarello
Dipartimento di Informatica
Università Ca' Foscari di Venezia
{auce, pitt}@dsi.unive.it

D. Fogli
Dipartimento di Elettronica per l'Automazione
Università degli Studi di Brescia
fogli@ing.unibs.it

P. Mussio
Dipartimento di Informatica e Comunicazione
Università di Milano
mussio@dico.unimi.it

Abstract

This paper discusses a model-based approach to the design of complex interaction environments like virtual worlds, mixed and augmented reality. The environment a user interacts with is seen as a virtual environment populated by virtual entities, created and maintained active by a program interpreted by the computer, which can be described by specifying the behavior of the population. The specification of the behavior occurs along three dimensions: 1) programming languages to specify system computations; 2) user activity languages to specify user activities; 3) perceptual languages to deal with the physical characteristics of the messages from the machine to the user. These dimensions define an interaction modeling space which constitutes the frame in which the virtual environment is specified.

1. Introduction

A model-based approach to the specification of interactive environments allows a designer to reason in a rigorous way about several aspects, such as: the environment itself, which must be clearly organized in order to allow a user to understand which are the interaction opportunities and how they are related each other; the interaction metaphor, which must be consistent with the user profile and with the application domain; the interaction dynamics, which must make the user aware of all the changes in the system status; the user interface design, which must help the user in finding information useful for his/her task; and obviously the application itself and its links with the components devoted to interacting with the user. A formal model of the interaction process can support design, implementation and validation activities by providing a reference theoretical background and automatic tools; also semi-formal approaches, often necessary due to the complexity of real world applications, are viable helps for designers.

This paper proposes a model-based approach for the specification of complex virtual interactive environments. It aims at building a reference model for environments characterized by three-dimensional metaphors, multimodal input and output, and navigation as primary interaction style. The model supports the designers during their tasks, as well as the final users, whose interaction should be eased by the presence of well-defined rules along which the design develops. Finally, it provides a background for the validation activity, since the expected behavior of the system can be described according to a rigorous description, hence can be compared with the actual observation of its evolution.

The approach generalizes the model of human computer interaction introduced by the Pictorial Computing Laboratory (PCL) [2], which is based on the concept of *characteristic pattern*. The generalized model accounts for entities in virtual environments which extend the concept of widget, common in WIMP interfaces. The characterizing features of this model (which will be called *PCL model* for brevity) lead to the definition of an *Interaction Modeling Space* (IMS) [1] in which each space dimension represents, at an adequate level of abstraction, types of languages: 1) programming languages to specify system computations; 2) user activity languages to specify user activities; 3) perceptual languages, which are languages devoted to deal with the physical characteristics of the messages from the machine to the user.

The PCL model and the interaction modeling space constitute the frame in which a *Virtual Interaction Environment* (VIE) is specified. The environment is defined in terms of the *Interaction Locus* model [3, 5], designed for structuring virtual spaces in which users can perform an organized series of interactive experiences. The requirements of the VIE are identified using the PCL model; then, the VIE is characterized in the Interaction Modeling Space, obtaining the specification of VIE computational and interaction characteristics.

Due to size constraints, in this paper the PCL model and the Interaction Locus model will be briefly surveyed, addressing the reader to the bibliography, and specifically to [2, 5] for the rationale and the details of the two models.

2. Extending the PCL Model to Virtual Interaction Environments

In the PCL model, the human computer interaction process is modeled as a sequence of cycles: in each cycle the human detects the events generated by the machine, such as a screen image or a sound, derives their meaning, decides what to do next, and manifests his/her intention by operating on the input devices of the system. The system perceives these operations as a stream of input events, interprets them, computes the response to human activity and materializes the results through its output devices.

The human interprets the machine output by recognizing *characteristic structures* (*cs*), i.e., sets of system generated events perceived as functional or perceptual units; the system plays the role of the second interacting entity, through a set of *application programs* which compute the system reaction to the user activity. The computer interpretation creates and maintains active a set of entities, that we call *virtual entities* (*ve*), which extend the concept of widgets and virtual devices, being more independent from the interface style and including interface components possibly defined by users at run time.

An interactive system appears to the user as an envi-

ronment constituted by virtual entities interacting one another and with the user through the input/output devices. The environment is obtained by a set of elementary virtual entities. Virtual entities can be recursively composed, from elementary to complex ones through suitable composition rules.

A virtual entity is created and maintained active by a set of programs *VEP* = ⟨*IOP*, *AP*, ⟨*IF*,*OF*⟩⟩, where *IOP* is the set of input and output programs materializing and maintaining the *cs* of the *ve* on some output device, and capturing user activity related to the *ve*; *AP* is the set of application programs computing the *ve* reaction to the user activity; *IF* and *OF* are the set of input and output functions that compute the inputs to *AP* by relating the input events produced by *IOP* to *cs*, and map the results of *AP* to commands for *IOP*. At each instant, the state of a *ve* is defined as a *characteristic pattern cp* = ⟨*cs*, *u*, ⟨*int*, *mat*⟩⟩, where:

- *cs* is the *characteristic structure* defining how the *ve* manifests itself to the user. It consists of a set of perceivable events which allow the user to evaluate the state of the *ve*. The events can be multimodal, i.e. visual, haptic or audio. The *cs* definition specifies the events that *IOP* has to generate in digital form;
- *u* specifies the state of the program *AP* being interpreted by the computer system to determine the current behavior of the *ve*;
- *int* is an interpretation function mapping the activities which can be performed by the user on the *cs* of the *ve* into *u*; *int* specifies the computation of *IF*;
- *mat* is a materialization function mapping *u* into the output manifestation of the *ve*, i.e. into the *cs*; *mat* specifies the computation of *OF*.

The global state of the interaction environment is described by a *multimodal sentence* (*ms*), a special *cp* whose *cs* is the whole multimodal message to be interpreted by the human and by the system. Coherently with the definition of a characteristic pattern, a multimodal sentence is specified as a tuple *ms* = ⟨*m*, *u*, ⟨*int*, *mat*⟩⟩, where *m* (message) is the whole *cs* perceived by the user, *u* specifies the current state of all programs whose execution rules the interactive environment, *int* and *mat* define the relations of elements of *m* with components of *u*.

Actually, *cp*s can be composed to form more complex ones. In the design of an interactive system, a finite set of equivalence classes of *cp*s, called atomic classes, and a set of rules for their instantiation and combination are defined from which more complex *cp*s, up to *ms*s, are derived.

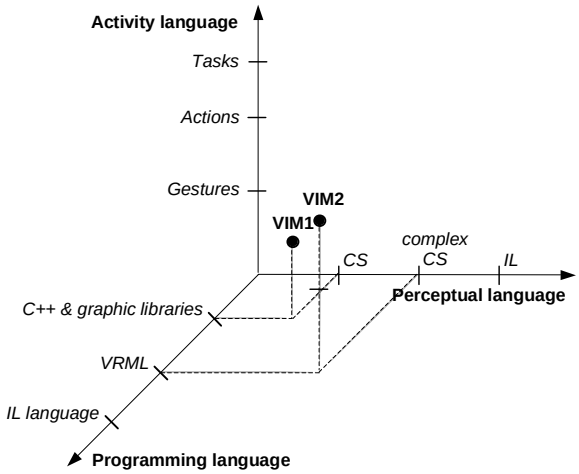

Figure 1. The Interaction Modeling Space for 3D interaction

In each interaction cycle, a multi-modal sentence ms_1 is transformed into a multi-modal sentence ms_2 as the consequence of some human activity *act*. A user activity *act* is specified as *act* = ⟨*op*, *cs*⟩, where *op* is the physical operation (or a set of physical operations) performed by the user, represented as digital events, and *cs* is the characteristic structure on which *op* is performed. The system relates *op* on the *cs* to a *cp*, and interprets it as a command from the user. Then it fires the consequent computation, referred to in the *u* associated to the *cs*, which often implies the change of the appearance of the *cs.*

3. *VIM*s in the Interaction Modeling Space

The interaction process was modeled in [1] within an *Interaction Modeling Space*. In this space a partially ordered set of interaction machines is represented, in analogy with the hierarchy of real and virtual machines used in computing systems. In modeling *ve*s, the sets of applications programs *AP* can be specified through languages at different levels of abstraction. In order to define the *ve* with reference to the interaction process, it is also necessary to define (a) the user activities *act* as perceived by the machine, which constitute the inputs determining the development of the computation in time, and (b) the output of the *ve* dynamics, i.e. the *cs*s which allows the user to decide which activity to perform. They can be described starting from a set of atomic elements to set up more complex ones. Therefore, we speak of "activity languages" [8] and of "perceptual languages" (a generalization of pictorial languages to multimodality) of *cs*s.

Since activities and *cs*s can be described at different levels of abstraction, close to the machine or user and task-oriented, we can define concrete and abstract interpreters of activities and *cs*s. The combination of program, activity and *cs* interpreters into one abstract or concrete machine is a *ve* generator, capable of interpreting the user actions and generating *cs*s. We call it a *Virtual Interactive Machine* (*VIM*).

Figure 1 shows an extension of the *Interaction Modelling Space* presented in [1] to multimodal interaction in a 3D space. On each axis, close to the origin, languages at the lowest level of abstraction are put. Each point in this space represents a set of *VIM*s, i.e. a set of hypothetical interactive systems defined by their activity languages, their programming languages and their perceptual languages. The levels on the programming language axis are specified using the languages currently used for programming 3D scenes. The *VIM*s in the set are different instances of the same functional behavior at that level of abstraction.

We can identify the sets of virtual machines used by the different categories of users that act in virtual environments. We shall focus our analysis on interaction through VRML browsers [7], that may be used for desktop virtual reality or mixed reality situations [4] where a

3D output is displayed on a PDA while the user moves in the real world.

The first level, *C++ & graphics libraries*, refers to languages used by skilled programmers; they use efficient languages and graphics libraries (such as OpenGL) that allow them to avoid direct references to low level constructs. A consistent trend aims at incorporating the implementation of those libraries into the hardware level of graphics cards, therefore they are not represented in the figure. *VRML* is positioned at a higher abstraction level; it implements constructs for geometry description and a set of interaction primitives for catching the user input and for guiding the modification of the 3D scene. Customized languages for 3D interactive scene authoring stand at the highest abstraction level. The *Interaction Locus Language*, which is described in Section 4, is located at this level.

Among the VIMs that we consider, the virtual interaction machine used by the developer of VRML engines for browsing 3D worlds (*VIM1*) is the closest to the real interaction machine, i.e., to the hardware. Developers use efficient programming languages and graphics libraries for developing VRML browsers. They use raw input coming from user gestures to define different categories of actions that will be used by authors of 3D worlds. For example, they may build the action *goto* extrapolating from the mouse movement the position associated to the final mouse click and using this position for changing the viewpoint on the scene, ignoring the intermediate mouse motions. Developers also use basic *cs*s (such as points and polygons) to derive complex *cs*s (such as cubes, spheres and cones). Therefore the virtual interaction machine *VIM*1 is specified by the coordinates ⟨C++ & graphics library, *cs*, gestures⟩.

At a higher abstraction level we place the *VIM* used by authors of VRML 3D worlds (*VIM*2 in Figure 1). They use a higher level language, VRML, to describe an interactive world, combining low level interaction primitives in order to build complex behaviors. From a communicational point of view, they aggregate simple audio and visual primitives to communicate more complex meanings, e.g., aggregating three boxes on the ground they obtain a portal. Concerning the activity language, they take user *actions* as input for the evolution of the 3D world. Therefore *VIM*2 is defined by the coordinates ⟨VRML, complex *cs*, actions⟩. *VIM*2 exists in the World Wide Web, and lies on the top of a well defined middleware system [6]. Its users can use distributed resources without being bored by low level management activities but being constrained by the grain induced by the middleware tools adopted.

VRML is one of the few available languages that allows programmers to specify both the geometry and the behaviors of the entities that populate the 3D interactive scene. Using the VRML language, an interactive sequence where the user is requested to click a door in order to open it can be programmed as a touch sensor associated to the door geometry; the touch event is sent to a script node managing the modification of the door geometry in order to open it. Such an abstraction level doesn't help authors with low level of expertise in VRML internals, that would be more comfortable programming the interaction sequence with constructs such as user.click(door) -> system.open.(door), possibly through a visual interface.

4. The Interaction Locus for high level authoring

VRML doesn't put any constraint about the location of the 3D scene where the action may take place. While this freedom can be useful in a few cases, in most situations this contrasts with the ordinary experience where actions take place in specific zones that have a morphologic identity and that are deputed to performing specific actions. To overcome such problems, some of the authors of this paper have introduced in previous works [3, 5] the concept of *Interaction Locus* (*IL*) as a means to structure a 3D scene in a number of well defined areas, characterized by specific morphological identities, and associated to the performance of specific actions. *IL*s typically are organized in sets and contain active objects the user interact with. The user experience is progressively built as a sequence of related actions happening in different areas. Both *IL*s and interactive objects are categorized in classes, like in real world situations where different locations and objects may belong to the same typology, such as the stands of a virtual fair, or the showcases in the fair pavillions. Actions themselves are categorized in classes and are selectively available to the user depending on the location he/she's currently navigating.

The introduction of the *IL* concept has a twofold nature, being useful both for formalization and for improving interaction. As a formal aid, it favors environment analysis, monitoring and transformation into lower or higher level descriptions. As an interaction aid, it increases, through a set of multimodal stimuli, the user awareness of the location where he/she's interacting, and can limit interface overcrowding by making available to the user only the types of actions that are appropriate for that specific *locus*.

The *IL* language, used for defining the 3D scene interactivity, is a high-level programming language that uses as constructs the *IL*, defined in terms of spatial boundaries and a set of multimodal attributes characterizing its identity. The *IL* language uses VRML constructs

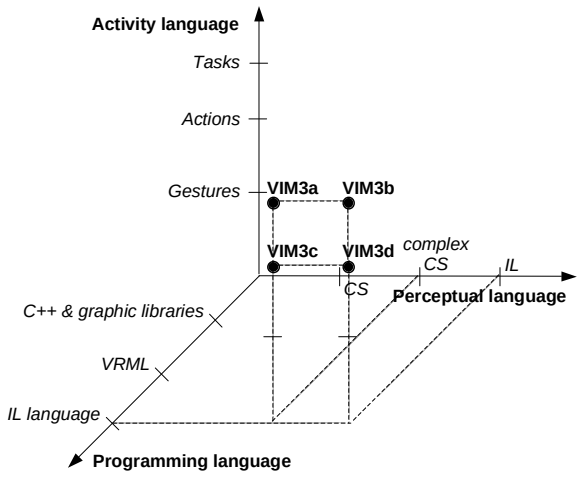

Figure 2. The VIMs for high level authoring

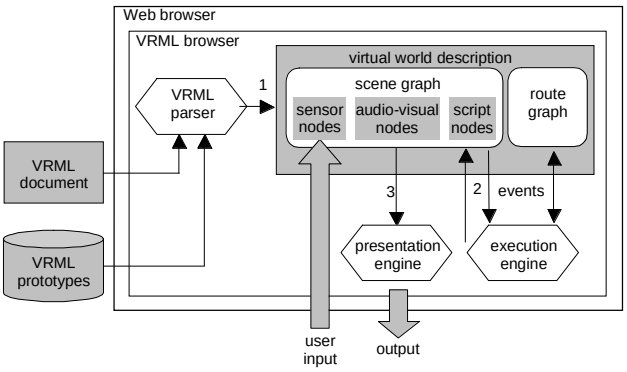

Figure 3. A VRML based architecture for *VIE*s

as building blocks. In particular, classes of *IL*s and actions can be built using their lower level programming constructs, and stored in repositories available to high level authors. With the *IL* language a high level author (the *IL* author) is enabled to organize the 3D experience in a set of locations suitable for interaction, and can specify for each locus which actions are allowed or forbidden.

While the *IL* author uses the *IL* language as programming language, he/she takes advantage of perceptual and activity languages at different levels of abstraction; therefore, as evidenced in Figure 2, the author uses *VIM* instances belonging to four different classes: *VIM*3*a*, *VIM*3*b*, *VIM*3*c* and *VIM*3*d*.

The *VIM*s used by the final user interacting in the 3D world belong to the same class of those used by the *IL* author. In fact, the final user faces with the same languages used by the author; he/she may use both actions and tasks for input, recognizes the complex *cs* projected on the 2D place because they represent objects available in the everyday experience, and may recognize the structure of the 3D world thanks to the presence of *interaction loci*.

5. Mapping virtual entities on the implementation architecture

To implement the model of *VIE* so far discussed, we propose an architecture based on a standard VRML browser, whose principal components are the *parser*, the *execution engine* and the *presentation engine* (Figure 3).

The VRML parser scans the VRML document describing the interaction environment and, optionally, external VRML prototypes, in order to build the scene graph (label 1 in Figure 3), which is a data structure representing the nodes defined in the VRML document: the nodes that can be perceived in the generated 3D world (audio-visual nodes), the sensor nodes for catching the user input and the script nodes for modifying the scene graph at run-time. The parser uses the routes described in the VRML document for building the *route graph* mapping the communication channels (i.e., the relations) among the different nodes of the scene graph. The internal representation of the virtual world is given by all these data structures.

During the interaction phase all the events generated externally by the user or by the nodes inside the 3D world are communicated to the execution engine (label 2 in Figure 3); the execution engine queries the route graph for knowing which events must be considered and modifies the scene graph accordingly.

Finally, the presentation engine renders the visual elements of the modified 3D scene on a clipped bi-dimensional surface (label 3 in Figure 3), renders the audio elements using the resources of the underlying hardware and manages interoperation with the host application (the web browser) for complementary presentation tasks, such as the visualization of web pages requested by the active components of the 3D world.

A correspondence exists between the components of the architecture described above and the program *VEP* implementing a virtual entity defined in Section 2:

- *IOP* is made by the presentation engine that materializes the virtual world on the screen and on the available sound devices and by the VRML browser tools which capture and digitize the user operations;
- *AP* is represented by the execution engine and the structures of the virtual world description that control its evolution (the script nodes and the route graph);
- *IF* is made by the scripts associated to the sensor nodes that catch the user input and map them into activities;
- *OF* is made by the set of programs producing and modifying the audio-visual nodes.

Establishing a suitable mapping between the implementation architecture of a VRML browser and the *ve* concept allows considering the VRML browser as a middleware that can be used for the definition of a superior layer presenting to authors and users a structured vision of a 3D interactive world. This layer takes advantage of the concepts of virtual entity and Interaction Locus, for building a more intuitive representation of the 3D scene, both for authors and users.

6. References

[1] P. Bottoni, M. F. Costabile, D. Fogli, S. Levialdi, P. Mussio. Multilevel Modelling and Design of Visual Interactive Systems, in *Proc. IEEE Symp. on Human-Centric Computing Languages and Environments*, Stresa, Italy, 256-263, 2001.

[2] P. Bottoni, M. F. Costabile, P. Mussio. Specification and Dialog Control of Visual Interaction, *ACM Trans. on Programming Languages and Systems*, 21(6), 1077-1136, 1999.

[3] A. Celentano, F. Pittarello. Observing and Adapting User Behavior in Navigational 3D Interfaces, in *Proceedings of AVI 2004, Working Conference on Advanced Visual Interfaces*, Gallipoli, Italy, 2004.

[4] P. Milgram, F. Kishino. A Taxonomy of Mixed Reality Visual Displays" *IEICE Transactions on Information Systems*, vol. E77-D (12), 1994.

[5] F. Pittarello. Accessing Information Through Multimodal 3D Environments: Towards Universal Access. *Universal Access in the Information Society Journal*, 2(2), 189-204, 2003.

[6] T. Plagemann. Middleware + multimedia = multimedia middleware? *Multimedia Systems*, 8, 395-396, 2002.

[7] Virtual Reality Modeling Language. ISO/IEC DIS 14772-1, http://www.vrml.org/VRML97/DIS, 1997.

[8] A.I. Wassermann, P. A. Pircher, D. T. Shewmake, M. L. Kersten. Developing interactive information systems with the user software engineering methodology. *IEEE Trans. on Software Engineering*, 12(2), 326-345, 1986