Transactions on Biomedicine and Health vol 6, © 2003 WIT Press, www.witpress.com, ISSN 1743-3525

Multilevel implementation of the dynamic virtual environment

D. Korošec, A. Holobar M. Divjak & D. Zazula
Faculty of Electrical Engineering and Computer Science
University of Maribor, Slovenia.

Abstract

Virtual environments (VEs) are nowadays often more than just static visualizations of 3D scenes in which user can navigate its avatar. From traditional applications in, for example, civil and machine engineering, 3D technology is moving forward to dynamic simulations and training systems. Unfortunately - there is still a lack of software standards that could be used to efficiently build such systems. VRML as probably the most standard technology for description of 3D scenes can add dynamic behaviour to 3D objects, but its implementation mechanisms, based on messaging, are rather crude. Three main levels exist: on the bottom simple deterministic behaviour can be hard-coded; medium level (scripting) then resolves many of the bottom level limitations; and finally, to create a complex system, separate, asynchronous applications connected to VRML form the third level.

In the paper we address this topic and demonstrate an example of medical training system for neonatal intensive care. The components of this system were created as several independent and interconnected dynamic 3D objects, whose behaviour was implemented over one, two or all three levels. We first explain them in details and demonstrate and clarify the benefits and drawbacks of each level with an example.

1 Introduction

Computer generated 3D environments can only capture a small part of the total complexity, dynamics and interactivity in the real world. Which part this is should mainly depend on application itself, but in practice the choice of the programming technologies may cause a significant bias. Unfortunately, no single standard solution exists today that would include all necessary building blocks for a complete interactive 3D application: description of 3D objects and rendering, background physical model, multi-user support, advanced interaction and immersion possibilities.

Virtual Reality Modelling Language (VRML) [1] as an open, platform-independent standard does offer some answers, especially for visualisation and simple animations, but beyond this one should use the VRML interfaces to scripting and general purpose programming languages - JavaScript, Java or C++. In systems built by using VRML and Java, the environment logic and behaviour of the objects is, therefore, divided among three implementation levels. One level is described in the core VRML using timer and interpolation nodes, which is usually the simplest and most deterministic behaviour. Second level consists of script nodes, where more complex functions can be specified. External programs and routines, communicating to the VRML scene graph and VRML rendering engine via External Scripting Interface (EAI) [7], add the third implementation level of such system.

In the next section of this paper a brief overview of the VRML is given first. Here we also present the basic animation mechanism using timers and interpolators, we call it Level 0. Its limitations can be resolved using scripting (Level 1) and combining separate applications (Level 2), which is both presented in Section 3. The architecture of our system - virtual baby application for neonatal resuscitation training - which uses techniques from all three levels is outlined in Section 4. Implementation of the respiration is shown as an example. In conclusions we summarise the advantages and drawbacks of each level and give hints on which level is appropriate for implementation of a particular system behaviour.

2 Virtual reality modelling language

Although the VRML is now almost an acronym for virtual reality, it was originally designed to hold static description of virtual worlds in VRML files [4]. The whole information about 3D objects and their properties is stored in a hierarchical structure called scene graph. Entities in the scene graph are nodes and consist of fields. With the appearance of version 2.0 (quickly turned into current version VRML97) [1], VRML can be used to perform simple animations and generate dynamic responses to the external happenings. This is supported through the message-passing mechanism, called event model, by which nodes in the scene graph communicate with each other [2]. Each node defines the names and types of events that may be generated (`eventOut`) or received (`eventIn`). Event paths between generators and receivers of events are called routes [3].

VRML browser, a program which interprets and presents VRML virtual worlds as if experienced from a particular viewing location, is often installed as a plug-in of a Web browser. VRML browser also provides a mechanism allowing the user to interact with the world through special nodes in the scene graph hierarchy called sensors. Sensors generate events in response to user interaction with geometric objects, navigation of the user through the world, or expiration of time. The essential receivers of these events are nodes that perform simple animation calculations. They are called interpolators and are usually combined with sensors or other nodes in the scene graph to make objects move. This *sensor⇒ interpolator⇒ geometry node* communication is the most basic method built into VRML standard to support animations.

2.1 Level 0: VRML animation cycle

For the purpose of our multilevel classification we denote the described mechanism Level 0. `TimeSensor` is its driving force: it stirs an interpolation node which provides an effective value to control the animated node. The example of such route is shown in Figure 1.

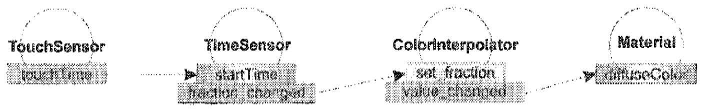

Figure 1: Schema of a basic animation cycle example - change of colour is triggered by clicking a touch sensor. `EventOut` fields are dark, `eventIn` fields are bright grey and exposed fields are iridescent.

Simple as it is, this method has some serious drawbacks. Most of them are **functional limitations**: only deterministic, periodic animations can be implemented. Also when started, the animation can not be interrupted. Awkwardness of this mechanism is also caused by some of the **inherent VRML limitations**: routes can only be set between named nodes, which makes anonymous communication or multicasting practically impossible. The same goes for interaction with dynamically added nodes.

But there is another problem related to event model itself - **performance limitation**. No control is possible over how often events are generated by each `TimeSensor` nor different priorities can be assigned to animation cycles of different importance. Some of the here mentioned limitations can be resolved using advanced animation mechanisms as described in the following sections.

3 Advanced animation mechanisms

3.1 Level 1: Scripting (SI)

Applications often require sophisticated program logic to control various pathways, check user input and access system resources directly. These demands contradict the principle of simplicity and universality of the VRML standard and are not supported in the main event model. To perform more complex, application-specific operations, the `Script` node was included (Figure 2). Using the rules of the Script Interface (SI), it can be used to implement a node with user-defined behaviour. The `Script` node has two main features:

- it can contain an arbitrary number of fields, `eventIns` and `eventOuts`;
- all received events are processed by a program, specified in the `URL` field.

When the `Script` receives an event, it calls a special method containing the user-implemented program. The actual name of the method depends on the programming language used. In Java, the method `processEvent()` [5] is called. In Javascript, the method has the same name as the eventIn which received an event [6]. Along with the actual value the timestamp is also received. The events generated by the `Script` method have the same timestamp as the event which triggered it. Script Interface does not limit the choice of programming language, but most VRML browsers support only Java and Javascript.

Transactions on Biomedicine and Health vol 6, © 2003 WIT Press, www.witpress.com, ISSN 1743-3525

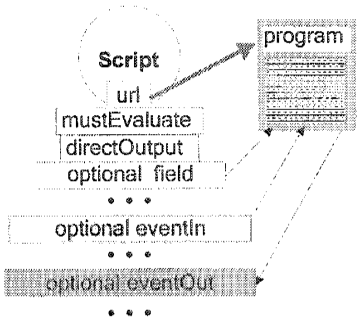

Figure 2: The `Script` node and its fields. The `URL` field contains a reference to the program code.

3.1.1 Scripting as a solution to functional limitations of Level 0

`Script` nodes can be used to greatly enhance the basic VRML animation mechanism, described in Section 2. To make the animation respond to various parameters, these parameters should be used as inputs to the `Script` node (as eventIns). That way the `Script` program is evoked as soon as one of the parameters is changed. By sending eventOuts it can change the state of the animation, modify the interpolation values, etc. The eventOuts should be routed to every node that we want to control. This way the `Script` node behaves as a supervisor node for the underlying mechanisms of Level 0. We can use it to select between different animations, to modify the length of the animation, to change interpolation values with time, to start/stop the animation, etc.

3.1.2 Scripting as a solution to performance control

One of the greatest weaknesses of the VRML standard is incomplete definition of the `TimeSensor` node. `TimeSensor` generates events as time passes and is a basis for all animations in the VRML world. To achieve the best performance, most VRML browsers generate time-related events as often as possible, and so decrease the responsiveness of virtual world. Another weakness of the `TimeSensor` node is the lack of the field for priority settings. All `TimeSensor` nodes in the scene generate events with the same frequency and there is no mechanism to slow some of them down. Most applications, however, consist of a few detailed animations and a number of less important time-dependant actions.

Therefore, we developed a prototype of a time-dependant node (we called it `Timer`) [9], whose frequency of generated continuous events can be limited. It has almost exactly the same program interface and behaviour as the standard `TimeSensor` node (except the frequency of continuous event). An additional field, called `delay`, which sets the minimum time interval (in seconds) between two consecutive continuous time events, was added to the `Timer` node. Setting the maximum frequency of each individual instance of the `Timer` node reduces the CPU load and ranks the instances of the `Timer` node in the virtual scene by their importance.

The `Timer` node is implemented as an instance of `Script` node combined with the Java code, and can transparently be included in any VRML scene. A Java thread is used to trigger its events, enabling quick responses to the requests from virtual world. Unfortunately, the execution of Java byte-code in Java Virtual Machine is still rather slow. As a consequence, our `Timer` node isn't capable of generating events with frequencies over 100 Hz.

3.2 Level 2: Connection to other applications (EAI)

An interface was developed to enable the external environment access to the nodes of the virtual world. It incorporates the highest level of generality and supports implementations based on individual programming languages, as well as implementations based on protocols. EAI allows three types of access to the VRML scene [7, 8]:

- sending events to `eventIn` fields of named nodes inside the scene,
- reading the last value sent from the nodes inside the scene and
- getting notified when events are sent from nodes inside the scene.

The schematics of an external application communicating with the VRML scene is depicted in Figure 3.

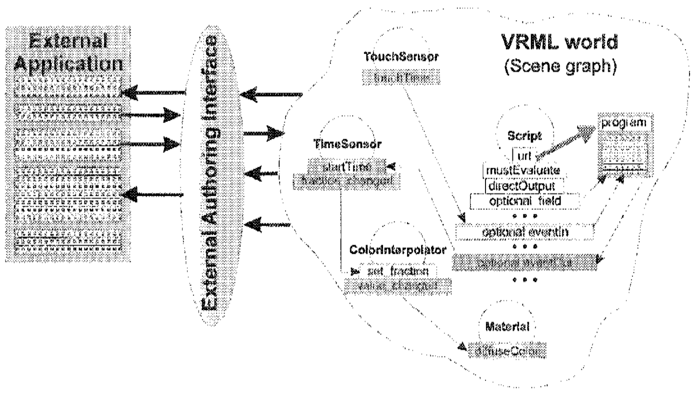

Figure 3: The connection of the stand-alone external application with the VRML world through the External Authoring Interface.

The main difference between SI and EAI is in the connection between the scene and the code of the user program. In SI, the program code is still under the browser control (`Script` programs are activated by receiving events). In EAI, the program runs independently from the VRML browser, which enables asynchronous execution of user program independently of the activity in the virtual world.

4 Complete virtual training case – neonatal intensive care

We have designed and implemented a prototype virtual environment for medical training in neonatal resuscitation (VIDERO). The central element of this environment is the dynamic virtual model (avatar) of a newborn child, built using VRML and Java [10]. Physiological variables relevant for training were chosen to be represented through the avatar: heart rate, respiration rate, skin colour and activity level (such as movement and crying).

Such computer–generated representation of a human being has one major advantage over the rubber doll, which is traditionally used in teaching neonatal resuscitation [11]: several vital signs can be dynamically rendered by generating their visual and audio representation. Students, working on an example case, are now faced with more realistic challenge – instead of listening to a verbal description of the baby's condition they have to continually asses visual and auditory clues from the baby itself and virtual devices around it.

In our model a teacher (mentor) remains the necessary key element: either he directly controls all observable parameters [12] (heart rate, respiration rate, skin colour, facial expressions and movements) of virtual baby or he predefines their trajectories over time by so–called scenarios. Based on actions the student chooses during training session (list of possible actions and their corresponding parameters are again described in advance by mentor), different scenarios can be triggered.

4.1 System architecture

VIDERO is a distributed application [13]. The system runs on a set-up of two connected personal computers and has an option to interface with the Polhemus motion tracking device and a head-mounted display. It consists of a mentor's and student's module that must run on two separate computers, of Replayer module and of several servers which are responsible for appropriate exchange of control data between the mentor's and student's modules, for recording and replaying the training courses, and for the communication with tracking devices. Structure of the VIDERO application is schematically depicted in Figure 4.

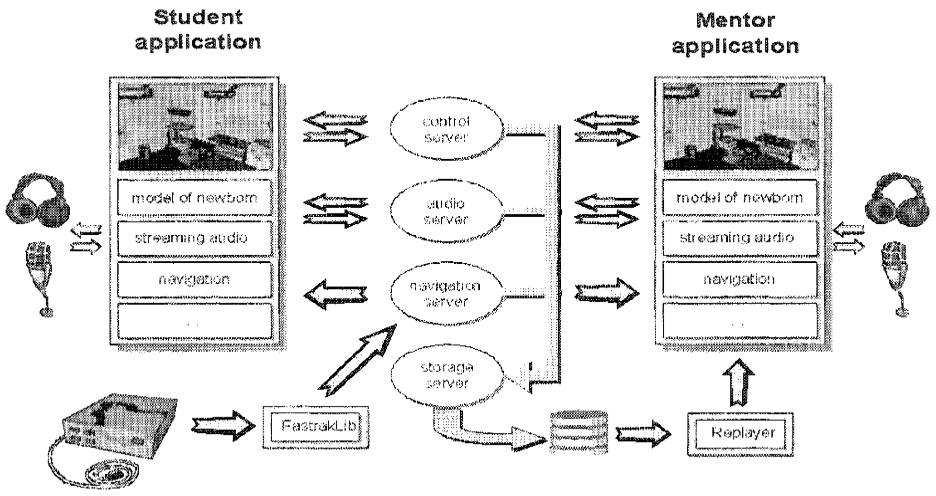

Figure 4: Modular structure of Virtual Delivery Room.

4. 2 Example of implementation: Respiration mechanism

One of the most important and investigated vital sings in neonatal medical treatment is the newborn's respiration cycle. Hence, much attention was paid to its implementation. It incorporates all three programming levels: Event model (Level 0), SI (Level 1) and EAI (Level 3) and can, thus, serve as a good example of how different simulation levels can interlace in VRML. The structure of the newborn's respiration mechanism is shown in Figure 5.

The respiration mechanism uses the standard VRML interpolators (Level 0) to animate the movement of newborn's chest (`BreathInCoordInterp` and `BreathOutCoordInterp` nodes in Figure 5), the audio clips of real newborn breathing (`BreathInAudio` and `BreathOutAudio` nodes), and the respiration curve on the virtual monitor (`BreathInInterpolator` and `BreathOutInterpolator`).

Four `Timer` nodes (Level 1) control the time progression of the respiration cycle i.e. dictate the durations of the different respiration stages: breath-in (`BreathInLength`), breath-out (`BreathOutLength`), and pauses after breath-in (`BreathInPause`) and breath-out (`BreathOutPause`), respectively.

The `BreathLooper` node is implemented using SI (Level 1) and serves as a central control node for synchronization of the different respiration stages. Using the EAI (Level 3) the commands from the external application (mentor's module) are passed to `set_transition eventIn`, carrying the information about the new breathing frequency and the time in which the transition to new frequency should complete. `BreathLooper` node then automatically interpolates the durations of each respiration stage in each repetition of the cycle and dispatches appropriate events to corresponding `Timer` nodes.

Similar mechanisms were used also for other vital signs. Their interpretations and visualizations are summarized in Table 1, along with their level of implementation.

Table 1: The interpretation and visualisations of different newborn's vital sings. Crosses denote the corresponding levels of implementation.

Vital sign	Interpretation, visualization	L0	L1	L2
Breathing	Chest movement	×		
	Sound	×		
	Graphical curve and current breathing frequency value displayed on virtual monitor	×		
	Interpolation of breathing frequency between successive respiration cycles		×	
	Control of breathing frequency over time			×
Heartbeat	Sound	×		
	Graphical curve and current heart rate value displayed on virtual monitor	×		
	Interpolation of heart rate between successive beats		×	
	Control of heart rate value over time			×

Transactions on Biomedicine and Health vol 6, © 2003 WIT Press, www.witpress.com, ISSN 1743-3525

Table 1, continued

Vital sign	Interpretation, visualization	L0	L1	L2
Colour of skin and lips	Skin and lips turn from normal to blue and back	×		
	Interpolation of the skin and lips colour		×	
	Control of the colours over time			×
Motion	Movement of the baby's head, arms and legs	×		
	Interpolation of the motion intensity		×	
	Control of the motion intensity over time			×
Cry	Replaying of the real baby crying audio clips	×		
	Facial mimics (lips, eyes and forehead motion)	×		
	Selection of the audio clips, synchronization of the moans with facial mimics			×

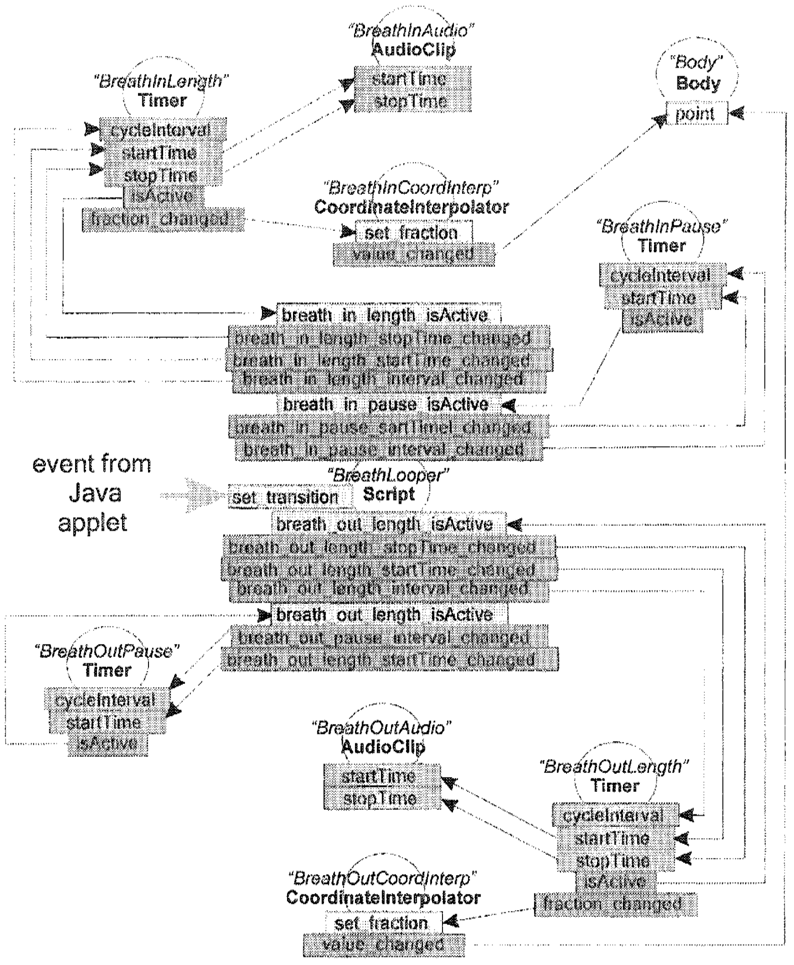

Figure 5: Schematic structure of the newborn's respiration mechanism. Circles depict different nodes with their names (cursive) and types (bold).

5 Conclusion

We demonstrated three levels at which dynamic behaviour of objects in virtual environments created using VRML can be implemented. As in all programming languages there is no single best implementation of certain behaviour, but it is important to know properties and limitations of each particular implementation level, as they are, based on our experience, summarised in Table 2.

Table 2: Properties of levels for implementation of dynamic behaviour in VRML.

Level	Advantages	Disadvantages
Level 0	Basic VRML mechanism Simple and standard Supported by all browsers	Functional limitations Inability to control browser performance
Level 1	Solves functional limitations of Level 0 Standard Supports several languages	Synchronous operation with respect to the VRML browser core Can respond only to triggered events Interferes with the performance of the rest of the VRML visualisation Does not support external fields
Level 2	Full functionality Asynchronous operation	Incompatibility of versions Not supported by all browsers No access to unnamed nodes No access to metainformation on the nodes No access to simple fields

Finally, in terms of appropriateness for certain functionality from the standpoint of ease of implementation and the rendering speed, the following guidelines can be given about each level: Level 0 implements elementary animations and deterministic periodical behaviour; Level 1 is suitable to model reflex behaviour, more complex deterministic and pseudo random animations; and Level 2 covers the cases, where VRML only provides visualisation, while computation of complex models and some other system tasks (multi-user communication, synchronisation, storage, etc.) run as separate processes.

To build complete dynamic virtual environments, as for example our presented virtual delivery room prototype, all three levels must be skilfully employed.

References

[1] The VRML Specifications, http://www.web3d.org/ VRML2.0/FINAL/Spec.

[2] Ames, A.L., Nadeau, D.R., Moreland, J.L. VRML sourcebook, John Wiley & Sons, Inc, New York, 1996.

[3] Hartman, J., Wernecke, J. The VRML Handbook: Building moving worlds on the web, Addison – Wessley publishing company, New York, 1996.

[4] Carey, R., Bell, G. The Annotated VRML 2.0 Reference Manual, Addison – Wesley Developers Press, Berkeley, 1997.

[5] The Virtual Reality Modeling Language: Java scripting reference, http://www.web3d.org/VRML2.0/ FINAL/spec/part1/java.html

[6] The Virtual Reality Modeling Language: JavaScript scripting reference, http://www.web3d.org/VRML2.0/ FINAL/spec/part1/javascript.html

[7] Marrin, C. Proposal for a VRML 2.0 Informative Annex: External Authoring Interface Reference, http://www.vrml.org/WorkingGroups/vrml-eai/ExternalInterface.html, November 1997.

[8] EAI design notes, http://www.vrml.org/Working Groups/vrml-eai/impl/design.

[9] Holobar, A., Zazula, D. Improved control of events in the VRML 2.0 application. Proc. of the 6th Euromedia conference, Valencia, Spain, pp. 67-71, 2001.

[10] Korošec, D., Holobar, A., Divjak, M., Zazula, D., Dynamic VRML for Simulated Training in Medicine. Proc. of 15th IEEE Symposium on Computer-based Medical Systems, Maribor, Slovenia, pp. 205 – 210, 2002.

[11] Halamek, L.P., Kaegi, D.M., Gaba, D.M., Sowb, Y.A., Smith, B.C., Smith, B.E., Howard, S.K. Time for a new paradigm in pediatric medical education: Teaching neonatal resuscitation in a simulated delivery room environment. Pediatrics, pp. 106-110, 2000.

[12] Bloom, R.S., Cropley C. et al. AHA/AAP Neonatal Resuscitation Textbook, American Heart Association, 1994.

[13] Divjak, M., Holobar, A., Prelog, I. VIDERO - virtual delivery room. Proc. of International Conference on Trends in Communications, IEEE Region 8 Student Paper Contest, Bratislava, vol. 1, pp. LIV – LVII, 2001.