VR-DeMo: a Tool-supported Approach Facilitating Flexible Development of Virtual Environments using Conceptual Modelling

Karin Coninx 1, Olga De Troyer 2, Chris Raymaekers 1, Frederic Kleinermann 2

(1) : Hasselt University,
Expertise Centre for Digital Media
and transnationale Universiteit Limburg
Wetenschapspark 2
3590 Diepenbeek, Belgium
+32 11 26 84 11
{karin.coninx,chris.raymaekers}@uhasselt.be

(2) : Vrije Universiteit Brussel
WISE research group
Pleinlaan 2
1050 Brussel, Belgium
+32 2 629 33 08
{olga.detroyer, Frederic.kleinermann}@vub.ac.be

Abstract: To improve the design of virtual environments, we have developed an approach, called "VR-DeMo", which allows virtual environments to be defined using conceptual modelling. The design of virtual objects with their behaviour is realized through domain concepts. Exploratory interaction design, with attention for usability, is facilitated through model-based development of interaction techniques and metaphors. This paper describes the development process and the tools supporting the integration of this approach in order to ease the developer's task. The realization of an interactive tool to model a park in its urban environment is presented as a case study.

Key words: high level modelling; conceptual modelling; model-based UI development

1- Introduction and related work

The development of virtual environments (VEs) is currently a technical and time-consuming process. Although a number of toolkits, such as VR Juggler [1] exist, developers often need to define the virtual environment and the interaction and behaviours within the virtual environment in a low-level programming language. This increases the number of errors that can occur and decreases the ease with which a VE application can be defined or adapted.

The VR-DeMo (Virtual Reality: Conceptual Descriptions and Models for the Realization of Virtual Environments) project aims to ease the development process of VE applications by defining applications on a higher level. These high-level models have the advantage to provide a basis for discussing the development of the VE application. Furthermore, by automatically translating some models into source code and interpreting other models at run time, the development time can be minimised. This also allows the designer of the virtual environment to flexibly change properties of the application.

There are a number of related research initiatives that also deal with the design of VR applications from a more high-level point of view. We review them briefly. The lack of high-level design methodologies for VR development has also been addressed in [2] with the presentation of VRID (Virtual Reality Interface Design). In this paper, four key components when designing VR interfaces are identified: object graphics, object behaviours, object interactions and object communications. The VRID methodology divides the design process into a high-level and a low-level phase and uses a set of steps to formally represent the environment. Although this methodology helps the designer to split the design into different steps and then refine them, it still does not allow the designer to express the design using his own terminology and relations.

James Willans et al. [3] have developed software that separates the process of designing interaction techniques from the process of building a specific virtual environment, making it easier for developers to design realistic interaction techniques and try them out on users. However, the way behaviours are being designed is still very much an engineering way and therefore, not intuitive for a non-engineer person. The approach taken by the VR-DeMo project is to make the design of the Virtual Environment more domain oriented and therefore, more intuitive for persons without engineering background.

The Rube methodology proposed by Fishwick et al. [4] facilitates dynamic multi-model construction and reuse within a 3D immersive environment. But this approach is still not that intuitive for a non-VR expert.

Kim et al. [5], describe a software engineering approach for designing Virtual Environments. This approach splits the design into three main components namely 'form', 'function' and 'behaviour'. They also extend the Unified Modelling Language (UML) for helping the designer to specify the design. Their approach is much more Software-Engineering design and therefore, the designer must have a good knowledge about software design,.

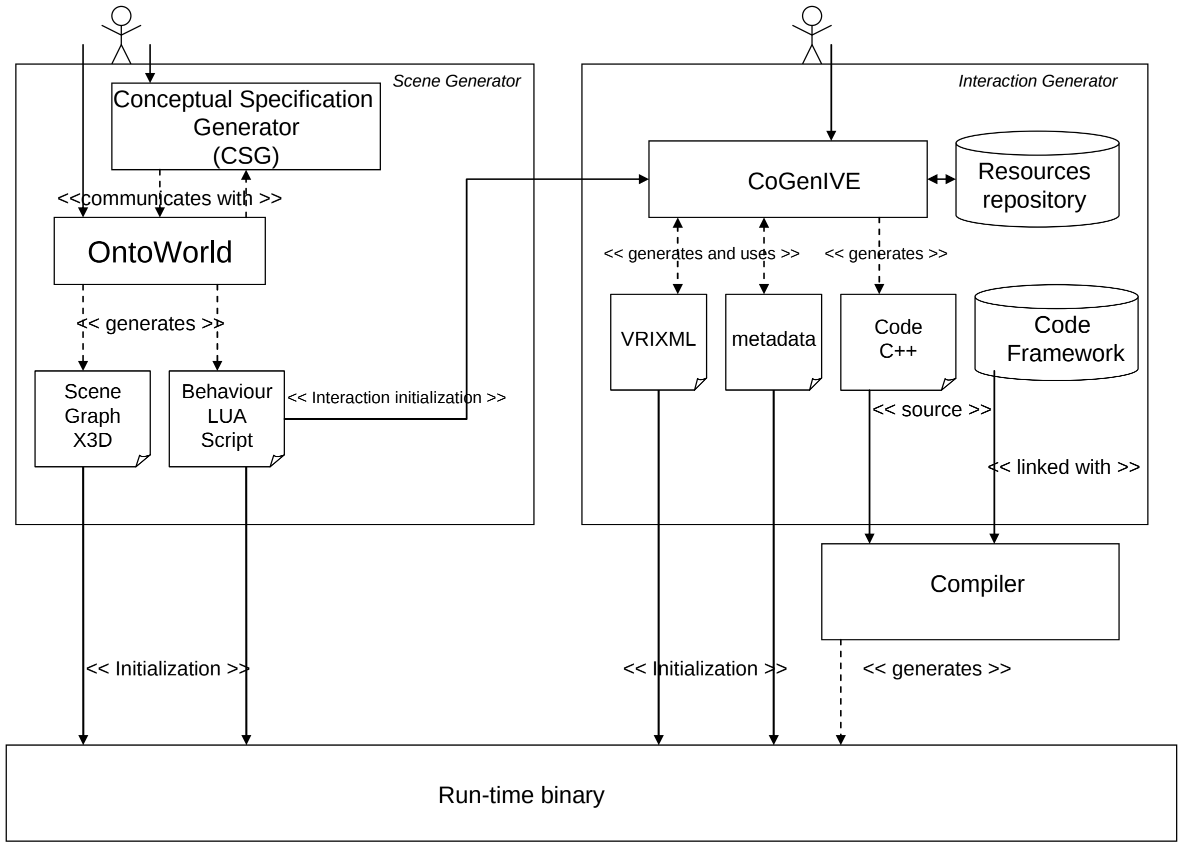

Figure 1 VR-DeMo approach

A commercial development environment that has similar goals as our approach is Virtools Dev [6]. Virtools is not intended to be a fully functional modelling environment. It only has some basic support to compose the virtual scene. Virtools also allows the designer to define behaviour for objects where behaviours can be designed graphically by combining a number of primitive building blocks. However, the function-based mechanism tends to be less comprehensible for novices. It also uses a graphical representation for behaviour (and interaction), which shows the execution flow, together with additional data-flow. We consider the approach taken by Virtools as more low-level than the one taken by the VR-DeMo project.

Furthermore, several models and description languages exist, which can be used to define user interaction. Examples are Petri nets [7], UML [8] and ICon [9]. Despite their focus on interaction, these models are very generic and are often cumbersome to use for describing interaction, particularly in virtual environments. Other models, such as ICO [10] and InTml [11], have been developed with interaction in virtual environments in mind. These models have the drawback that they are not easy to apply in a cognitive modelling approach, where the specified models have to be interpreted at runtime by the application. The Marigold toolset [12] can be compared to the VR-DeMo approach for describing 3D interaction. However, the flownets onto which this toolset is based can currently not be executed at run-time. Similarly, on top of the Cameleon framework [13] and the UsiXML process for defining context-sensitive user interfaces [14], a method has been created for designing 3D user interfaces [15]. However, this method needs further experimental validation.

This paper elaborates on the VR-DeMo approach. First, the general way of working when specifying and implementing a virtual environment is sketched. Next, the models of the different parts of a VE application are defined. Also, the tools that support the VR-DeMo approach are explained. Finally, we will detail a case study, an interactive tool to model a park, and explain how this case study is realized using the VR-DeMo approach and tools.

2- VR-DeMo approach

The VR-DeMo approach is based on a number of high-level descriptions and conceptual models. These can be divided into two logical parts, as depicted in figure 1. The first part, the scene generator (left side of the figure), consists of the definition of the virtual environment and the behaviour of the objects within. The other part, the interaction generator, defines the user interaction (right side of the figure). Both logical parts are elaborated on in section 3.

On top of the VR-DeMo approach, two tools have been created that translate the models and descriptions into source code and resource files that are used by the resulting application. Section 4 explains how these tools work.

As object behaviours can be triggered by user interaction, the interaction generator also uses some of the files generated by the scene generator. For instance, a light switch object can be instrumented with the behaviour description of turning a light on and off. When a user pushes the light switch, this behaviour must be invoked.

3- Modelling of virtual environments

Virtual objects, behaviours of these objects and the interaction with the user are important points of view on a virtual environment. Therefore, the design of models and descriptions for defining these aspects is a central activity in our research. The next subsections explain how VE applications can be defined with these models and descriptions and how they relate to each other.

3.1 – Modelling of virtual worlds

In VR-DeMo, ontologies are not only used to incorporate domain knowledge but they are at the same time used as underlying representation formalism, i.e. to capture the different description and models. The ontologies are represented using OWL [16]. To model the virtual world, the designer needs to perform two steps namely the *specification step* and the *mapping step* (see figure 2).

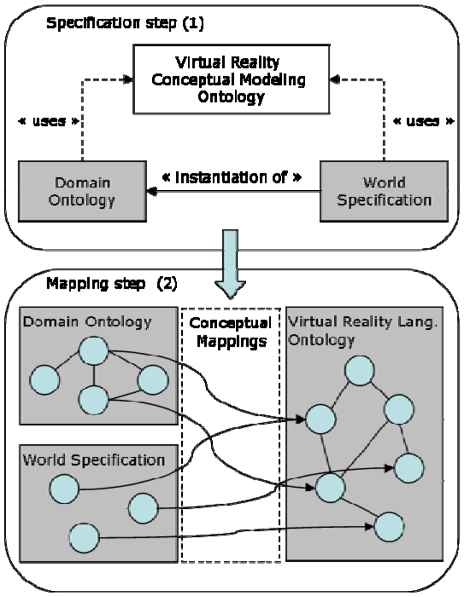

Figure 2 Modelling the virtual world

- **The specification step** allows the designer to specify the Virtual World at a high level using application domain knowledge and without taking implementation details into account. In the specification step, two levels are considered namely the domain level and the world specification level.

  At the domain level, the designer will specify the different concepts of the domain of which instances are needed in the VE. Concepts are comparable to object types in OO-design methods and they represent the different type of concepts available in the domain under consideration for the application. The concept specifications are maintained in an ontology called the *Domain Ontology*. This ontology describes the domain concepts by means of their (real-life) properties and their interrelationships. For example, for the case scenario (see section 5), this ontology would contain concepts such as Building, Tree, and Fountain. For each of these concepts, the ontology will also contain properties such as colour, material, depth, height, and width. These are properties directly relevant for the creation of the VE. Note that VR-DeMo can also import an existing ontology of concepts originally created and used for other purposes. Furthermore, for each concept, additional relevant semantic properties and information can be specified [17]. This extra semantic information can be used, later on, to add semantic annotations to the virtual world and to allow at run-time to search for locations in the virtual world based on these semantic annotations.

  The world specification level deals with the actual conceptual description of the virtual world to be built. A second ontology called the *World Specification* is used to maintain the specifications of the objects that will populate the virtual world. This is done by instantiating concepts described in the Domain Ontology. These instances represent the objects from the virtual world. For instance, in the scenario case (see section 5), for the concept Bench we will create multiple Bench-instances. An instance inherits the properties of its concept; but the values for the different properties must be specified here (if the default is not appropriate). Also information about the object’s location and orientation must be given, as well as information specific for the world itself (e.g., gravity, lights…). Like for the domain level, extra semantic information can be added to each instance beside the semantic information already inherited from their concepts.

  To define the domain level as well as the world specification, a number of high-level modelling concepts are provided, e.g., spatial and orientation relationship that can be used to position and orientate an object relative to other objects. For some type of users, this is a more intuitive method than the use of Cartesian coordinates. Furthermore, a graphical notation has also been developed to support the specifications. Details on this graphical language can be found in [18-20].
- In **the mapping step** the designer will specify how concepts and instances should be represented visually in the virtual world. Following the separation between the domain level and the world specification, there is the domain mapping and the world mapping.

  The domain mapping defines the mappings from the concepts in the Domain Ontology to VR implementation primitives. The purpose of this mapping is to specify how the instances of a domain concept should be represented (by default) in the virtual world. For example, a simple rectangular building could be mapped onto a VR primitive box. The low-level VR concepts that can be used in the mappings are described in an ontology called the *Virtual Reality Language Ontology*.

  Although objects are instance of a certain concept, different instances of the same concept may, in some case, require different representations. For this reason, the designer can override the default mapping of an instance inherited from its concept. This is done in the *World Mapping*.

3.2 – Modelling of object behaviour

Once the specification step and the mapping step have been done (see section 3.1), the designer can then model and attach behaviours to the different objects populating the virtual world. With the VR-DeMo approach, behaviours are defined independent of the structure of the object, and also independent of how the behaviour will be triggered. This improves reusability and enhances flexibility as the same behaviour definition can be reused for different objects and/or can be triggered in different ways. Furthermore with the VR-DeMo approach, behaviours are modelled through the use of a graphical notation [19-20]. The modelling of behaviour is done in two separate steps: (1) the *Behaviour Definition* and (2) the *Behaviour Invocation*.

- The first step consists of building *Behaviour Definition* Models that allows the designer to define different behaviours. To achieve this, a number of modelling concepts are introduced.

  (1) **An actor** can be seen as a kind of abstract object that is used to specify behaviour, instead of the actual object. An actor is used since we want the definition of a behaviour to be separated from the actual definition of the structure of the object.

  (2) **A behaviour** is defined for an actor. We distinguish between primitive behaviour and complex behaviour. In terms of primitive behaviour, there is: *move*, *turn* and *roll*. These primitive behaviours either change the position of an object or its orientation. The *move* behaviour can be used to express a change in the position of an object. To express a change in the orientation of an object, the *turn* can be used to express a rotation of the object around its top-to-bottom axis while the *roll* is used to express the rotation of an object around either its left-to-right axis or its front-to-back axis.

  To deal with modelling structural changes of objects, we also provide a number of pre-built complex behaviours: *transform, disperse, combine, group and ungroup*. More on these pre-built complex behaviours can be found in [20].

  In order to cope with more complex situations, both the primitive behaviours and the complex behaviours may be associated with a (textual) script.

  (3) **Operators** are used for modelling more complex behaviours. Operators allow combining simple and complex behaviour to form new complex behaviour. There are three types of operators: *temporal*, *lifetime* and *conditional operators*. The temporal operators allow to synchronize behaviours (example temporal operators are *before*, *meets*, *overlaps*, *starts*, …). Lifetime operators allow to control the lifetime of a behaviour (example lifetime operators are *enable*, *disable*, …) and the conditional operator allow to express a flow of behaviours by means of conditions.
- The second step in the behaviour modelling process consists of creating *Behaviour Invocation* Models. The *Behaviour Invocation* step allows attaching the behaviours defined in the Behaviour Definition step to the actual objects, and to parameterize them according to the needs.

As for the Specification step (see section 3.1), a graphical notation has been developed for the modelling of the behaviours. For more details see [19- 20].

The main advantage of the VR-DeMo approach for designing behaviours is that the designer does it more from a high-level and he is not concerned with the low-level implementation details. Furthermore, the VR-DeMo approach provides a generic way in the sense that once a behaviour has been defined, it can then be attached to different objects populating the virtual world.

3.3 – Modelling of interaction

Once the designer has modelled the virtual world and the different behaviours, the interaction can then be defined. The VR-DeMo approach uses model-based user interface development (MBUID) in order to define this. First, the tasks that the user can perform in the application and the task that the computer must execute accordingly are denoted using the ConcurTaskTrees notation [21], which orders these task in a hierarchical tree with time dependencies. This model is then used to define the interaction between the user and the system. Two different modes of interaction are possible: dialog and menu-based interaction and direct manipulation.

As we are developing VE applications, the menus and dialogs are also positioned in the VE itself. We have therefore created a description language, called VRIXML [22], which is able to describe these menus and dialogs and defines which events must be fired when the user interacts with them. These events can pass through the application in order to fire a certain functionality, some events, however, are intercepted by the VRIXML renderer. One of the events for instance indicates that a particular dialog must be shown. This allows the application developer to abstract away from the different menus and dialogs, which are defined by the user interface designer. Figure 3 shows an example of a dialog that has been activated by a menu. In this figure, the visual appearance of the menu is faded in order to underline that the dialog has become active.

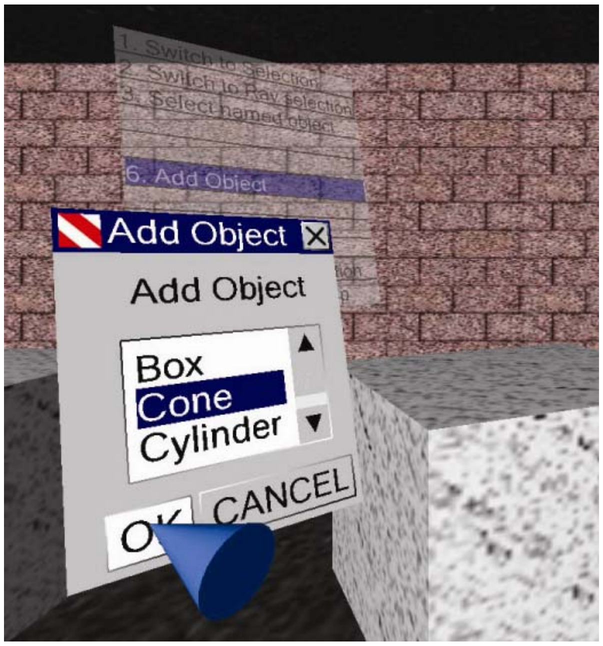

Figure 3 Example VRIXML elements

Since direct manipulation is the dominant interaction style in virtual environments and since existing interaction descriptions and notations do not suffice in order to describe

the rich interaction in virtual environments [23], we have created a notation for describing interaction in virtual environments, called NiMMiT, Notation for Multi-Modal Interaction Techniques [24]. This notation combines a state-driven approach with an event-driven and a data-driven approach. The states in a NiMMiT diagram define which parts of the interaction can be activated, while the actual activation depends on the events that are fired by the system. These events can be generated by the above-mentioned menus and dialogs or can indicate the user has moved the input device, has pressed a button or has used speech input. Finally, we can share data throughout a NiMMiT diagram in order to be able to save and retrieve certain state information. For instance, the current position of the virtual camera can be stored when beginning a travel technique. By restoring this camera position while finished this technique, the user can automatically return to the original position.
Finally, it is possible to hierarchically reuse a NiMMiT diagram in another diagram. For instance, the VooDoo dolls manipulation technique [25] can make use of different common selection techniques in order to select the objects to be manipulated. It is not necessary to redefine these selection techniques. They can be reused in the VooDoo dolls NiMMiT diagram.
Figure 4 shows how the virtual pointer selection technique can be defined using NiMMiT. In this simple example, collision detection is performed when the user moves the pointer. This is repeated as long as no collision is found. When collision with an object is detected, this object is selected and the selection technique is finished.

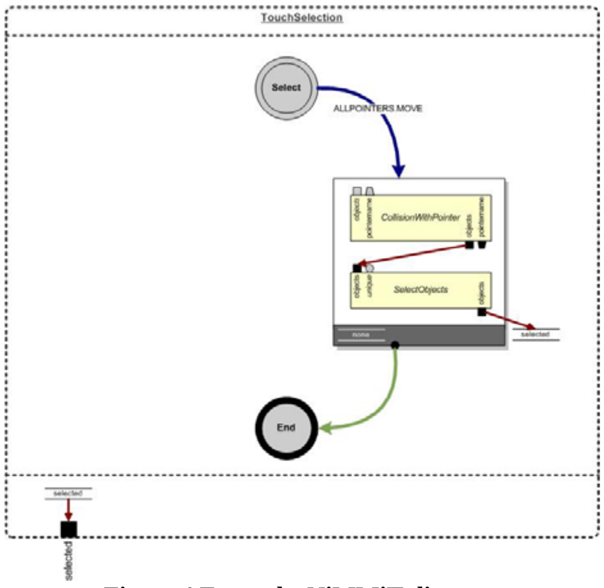

Figure 4 Example NiMMiT diagram

One of the additional advantages of the NiMMiT notation is that it abstracts away from the devices that are actually used. Each possible device that the user wants to use must generate a number of events, which can be understood by the NiMMiT interpreter. In our current implementation, we realize this abstraction by using VRPN, the Virtual Reality Peripheral Network [26], in order to address as many devices as possible. This library abstracts the implementation of different devices away. The events generated by VRPN are translated into NiMMiT events and collected, together with VRIXML events, in a multi-modal melting pot [27]. For further information and more examples, we refer the interested reader to [24].

4- Tool support

As can be seen in figure 1, the VR-DeMo approach is realized through two tools: the scene generator and the interaction generator. This section explains how these tools can be used for modeling VE applications.

4.1 –Scene Module

The *scene generator* deals with the design of the virtual worlds (with its objects and their behaviours) based on the approach explained in section 3.1 and 3.2. This module has two sub modules.
The first sub module is the *Conceptual Specification Generator* (CSG) and is a graphical diagram editor supporting the modelling of the behaviour and the specification of the static scene. This module has been implemented as an extension to Microsoft Visio [28]. It supports the graphical notation that has been developed for the specification step (section 3.1) and the modelling of the behaviour (section 3.2). The graphical elements can be dragged and dropped onto a canvas and connections can be made. Properties can be added, displayed and modified by double-clicking on the graphical elements. The CSG is a graphical interface for the specification phase. The information collected (specifications) using this graphical interface will be passed to the second module, *OntoWorld*, which will store it in its ontologies. Consistency between the diagrams and the ontologies in OntoWorld is maintained automatically. OntoWorld can also import existing ontologies (written in OWL) that define the concepts at the domain level.
OntoWorld also comes with a library of predefined VR-objects which can been modelled with other modelling software package like 3D Studio Max. This library is used to support the Mapping step. Concept and instances defined in the specification step can be mapped directly on these VR-objects. OntoWorld also provides a way to preview the static virtual world. Furthermore, it also allows attaching additional semantic information to the different concepts and instances as described in section 3.1.
Based on the models and specification provided, the scene generator module generates three files. The first file represents the scene with the objects together with their behaviours. The file is generated in X3D format [29]. The second file contains the behaviour using LUA [30] script and the third file contains the semantic annotation specified during design-time and is given in an MPEG-7 format [32]. The X3D file and the LUA script are then given to the to the second tool the *interaction generator* in order to attach interaction to the virtual world.

4.2 – Interactivity

To support the development of the user interaction with the models as defined in section 3.3, the interaction generator uses a tool called CoGenIVE, Code Generation for

Interaction Virtual Environments. This tool allows the user to draw the different parts of the models. While a NiMMiT editor will be integrated in CoGenIVE, the first validation has been done with a Visio extension. Menus and diagrams are visualized using the OpenGL-based VRIXML renderer, which is also used by our application framework. Hence, the designer can see these menus and dialogs as they will be visualized in the final application (unless the application uses a different renderer for VRIXML).

CoGenIVE not only allows the designer to define the different models, but also allows to make connections between these models. First, the designer must define the task model using a ConcurTaskTree (CTT), as explained in section 3.3. From this CTT model, a dialog model is generated by using the algorithm from Luyten et al. [32]. This dialog model indicates when the different tasks can be executed.

Next, the designer must indicate when the different menus and dialogs must be shown by linking them to the different "states" in the dialog model. Furthermore, the designer can indicate when the different NiMMiT diagrams must be executed.

When the designer has finished modelling the interaction, CoGenIVE generates the necessary XML files in order to represent the NiMMiT diagrams and the VRIXML elements. The CoGenIVE tool also generates the VE application based on the VRment, our VE framework, and a number of code templates. The application developers only need to code the specific functionality in a number of predefined places and to compile this code. If certain code files need to be regenerated because the designer has added or changed the functionality, the manual changes are automatically taken into account by the CoGenIVE tool.

5- Case study: a city park designer

In order to assess the validity of the VR-DeMo approach and the usefulness of the tools, we have elaborated on a case study: a city park designer. This application allows users to design a virtual city park. The user of this application can, in a first phase, design the park by positioning objects, such as buildings, and statues in the virtual environment.

When the virtual city park is designed, the user can simulate certain behaviours. For instance, the different cars in the virtual environment will move about. A simulation of the sun has also been added, which allows the user to define the current time of the day in order to view how different shadows are projected. Alternatively, the current time of the day can also progress, resulting in an animation of the shadows. This section will elaborate on the different models used to define the application in relation with the VR-DeMo approach as explained in section 3.

5.1 – Definition of the virtual world

To define the virtual world (see section 3.1), the designer must start to specify the specification step and the mapping step.

The specification step starts with the domain level where the different domain concepts with their properties are specified. For the virtual city park, we have domain concepts such as *Building*, *Pine Tree*, *Bench*, *Park Block*, *City Block*, *RoadSign*, *Streetlight*, and *Fountain*. These concepts have properties. For instance, the concept *Pine Tree* has the properties: *name*, *diameter*, *height*, *type*, *material*, *colour*.

The world specification level is then used for instantiating domain concepts. In the case of the virtual city park, the concept *Pine Tree* is instantiated several times in order to form the city park. Then, values are specified for the properties (or defaults are used). For example, an instance of *Pine Tree* has been given a value of 10 meters for the height and a value of 1 meter for the diameter. It is also necessary to position the instances inside the space representing the virtual city park. Next to the traditional way of positioning instances by means of coordinates and to orientate them by means of angles, these instances can also be positioned using high-level concepts which position and orientate them relative to other instances (see section 3.1). In the case of the virtual city park, we can state that instance "pinetree1" is positioned left of the instance "birchtree2" with a distance of 5 meters. It is also positioned above the ground. Using *Conceptual Specification Generator* (CSG), the designer can specify this relation in a graphical notation way (see figure 5).

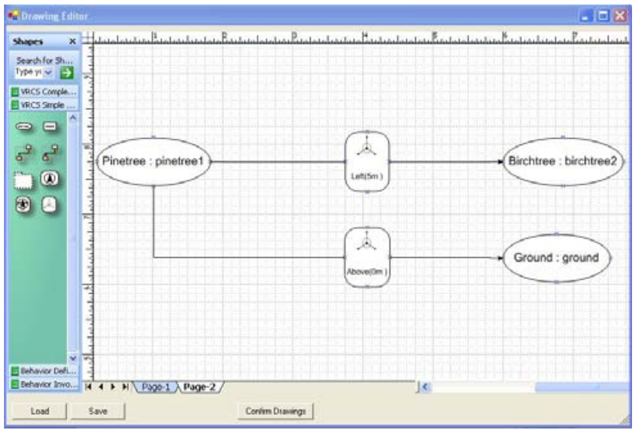

Figure 5. Specifying spatial relations using the CSG editor

Once the specification step has been finished, the designer should specify how the different instances should be represented inside the virtual city park (mapping step). Firstly, default mappings are specified for the different domain concepts. This is done by mapping domain concepts onto VR-objects (pre-defined in the VR Language Ontology). Concept properties must be mapped onto properties of the VR-objects. For instance, the concept *Pine Tree* is mapped onto a VR-object VR-Tree with properties height, width and depth. The properties of *Pine-Tree* are mapped as follows:

VR-Tree.width = pine-Tree.diameter
VR-Tree.height = pine-Tree.height
VR-Tree.depth = pine-Tree.diameter

5.2 – Definition of object behaviour

Once the specification step and the mapping step have been done, the virtual world at this stage is only described from a static point of view. In order to make it dynamic, behaviours must be modelled. To do this, the designer must model different behaviours and attach them to the different objects populating the virtual city park. Using the CSG editor, the designer starts by making *Behaviours Definition Diagram* for each of the behaviours that he wants to have in his virtual world. In the case of the virtual city park, there will be different behaviours. For instance we want the fountain to squirt water (the *Fountain behaviour)*; the sun should appear and disappear (the *Sun behaviour*); the street light should go on once the sun disappear (the *street light behaviour)*; and the bus should move and stop at different bus stops (the *bus behaviour*). To give an example on how the designer models a behaviour with the VR-DeMo framework, an illustration is now given with the bus behaviour.

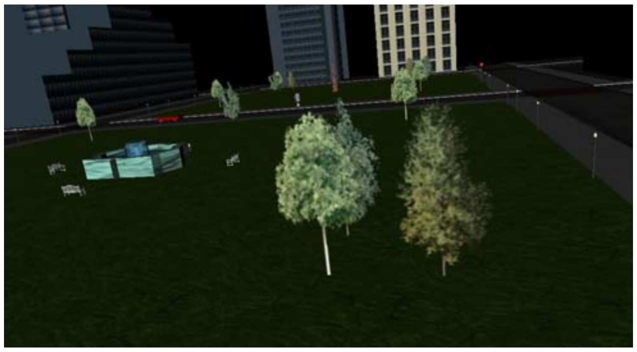

Figure 6 View of the Virtual City Park where the instance of "pinetree 1" is positioned left of the instance "birchtree2" with a distance 5 meters.

The bus behaviour makes the bus to move, stop for few seconds and turn. This behaviour will allow the bus to follow different routes. In our case scenario, this behaviour is called *Route66*. The behaviour *Route66* makes the bus at a certain position moving forward (run) for 25 meters, then stopping for a minute. After a minute, the bus runs again for another 25 meters, then turns right and it moves again for 50 meters. After 50 meters, it will then turn left and move forwards for 25 meters where it will stop for a minute. After a minute, it will run for another 25 meters.

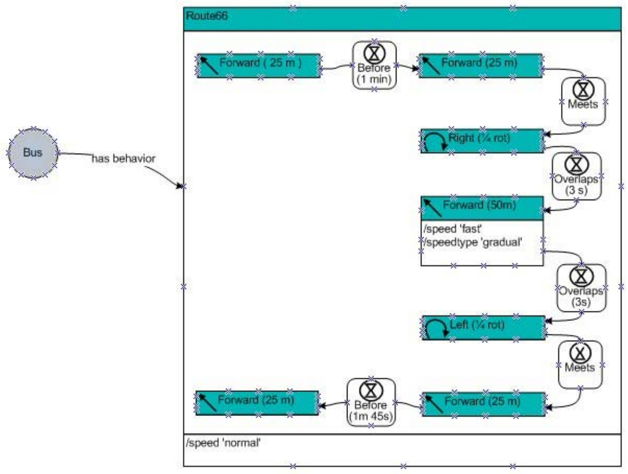

Figure 7. Behaviour Definition Diagram for the Bus behaviour Route66

To model this sort of behaviour using the VR-DeMo framework, the designer will use the CSG editor to define a *Behaviour Definition Diagram* (see section 3.2) for that behaviour. When creating a *Behaviour Definition Diagram*, the designer will first create an actor. Here, the actor will be called *Bus*. The designer will then state that this actor has a behaviour called *Route66* which represents the behaviour of the Bus mentioned above. It is composed of different time operators (*before*, *meets* and *overlaps*) combined with the primitive behaviours *move* and *turn* (see section 3.1) in order to make the bus moving, stopping and turning. Figure 7 shows the graphical specification of the Bus behaviour done with the CSG editor.

Once the behaviour has been specified by a *Behaviour Definition Diagram*, the designer will then determine to which objects (instance) populating the virtual world, this behaviour is attached to. To do this, he will create a *Behaviour Invocation Diagram* which will allow him to attach a behaviour to an object and which will also allow him to specify how this behaviour will be invoked. Again with the VR-DeMo approach, this will be done through the CSG editor. In the virtual city park and for *Route66*, the designer will attach the bus behaviour *Route66* to the instance object bus called *bus_nr_41* using the *Behaviour Invocation Diagram* shown in figure 8.

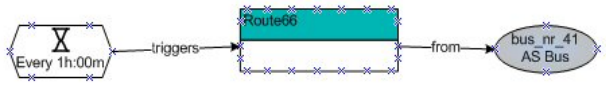

Figure 8. Behaviour Invocation Diagram for the Bus behaviour

In that figure 8, it can be seen that the behaviour *Route66* is triggered every hour for the instance *bus_nr_41*. The designer will do this sort of diagrams for each behaviour. Once the virtual world with its behaviour has been modelled, interaction can be attached.

5.3 – Definition of interaction

Interactivity in the city park designer application is supported by means of a spacemouse, a six degree-of-freedom input device. However, as the interaction is modelled in NiMMiT, which abstracts device details away, this device can easily be replaced by another one.

The interaction within the city park designer application can be divided into two phases. During the design phase, direct manipulation is the dominant interaction style. Users can add different objects, such as trees to the virtual city park. Figure 9 show the dialog containing the objects that can be created in the virtual world.

The objects in the environment can also be selected in order to be deleted or repositioned. A virtual hand is currently used, as this kind of interaction technique is well suited for a spacemouse. If another input device would be used, this can easily be changed to another metaphor by applying another NiMMiT diagram. For instance, the Go-Go interaction technique [33] could be used when working with tracked gloves.

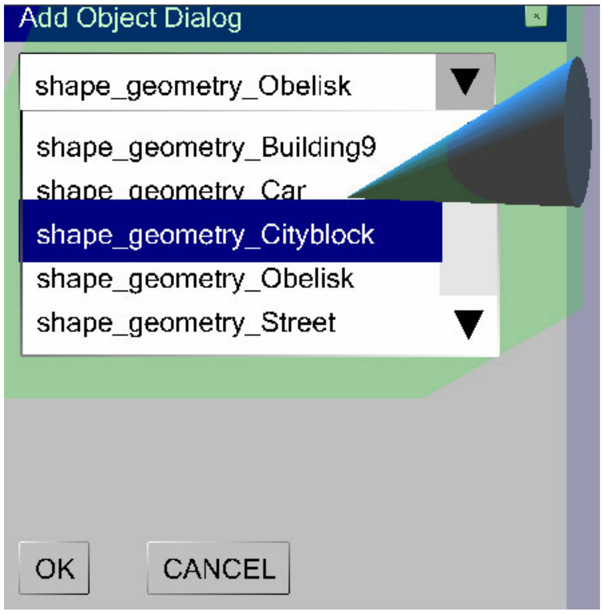

Figure 9 Dialog for adding objects

The user also needs to be able to scale the objects. Otherwise, all objects of the same type would have the same size. It is not desirable, for instance, that all trees have the same height. If the user has a second input device, two-handed scaling [34] can be applied. With this technique, the scale of an object depends on the distance between the user's hands. If the user moves his/her hands further away from each other, the currently selected object grows bigger. Likewise, an object can be shrunk by moving both hands together. Figures 10 and 11 shows an object before and after it was scaled using this technique.

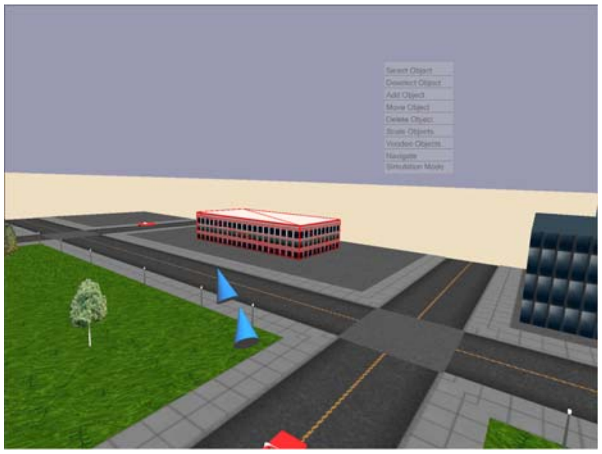

Figure 10 Two-handed scaling: before

Of course, the user of the city park design application is also able to move around within the environment. For this purpose, the flying vehicle metaphor [35] is used. We can distinguish two modes of using this metaphor. The user can either freely moves about the virtual city park in order to fully inspect it or can restrict his/her motion to exploratory walking above the ground plane.
During the simulation, the user cannot change the appearance of the virtual world anymore. Navigation, on the other hand, is still possible. The user can still use the navigation described before. Furthermore, when the user comes near certain points of interest, a corresponding behaviour is triggered from this interaction. For instance, when the user approaches the fountain, the *fountain behaviour* is activated. Likewise, a statue will give some information about itself by playing an audio fragment when the user approaches it.

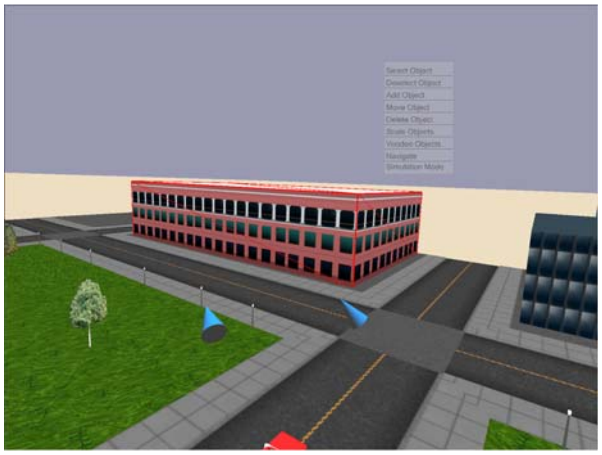

Figure 11 Two-handed scaling: after

Before the simulation is started, the user must fill-in some parameters. If the user wants the see a simulation of the sun and the corresponding shadows, the current time of day must be indicated along with the speed of the simulation. Also, the user must indicate if certain behaviours, such as the bus behaviour must be active. These parameters can be set by means of dialogs defined in VRIXML.
Furthermore, during the whole run of the designer application, some menus can be accessed in order to set parameters or to change the design phase to the simulation and back. It is not required that a 3D device, such as a spacemouse is used for this task since the city park designer also supports speech input.

6- Conclusions

This paper elaborated on the VR-DeMo approach for flexibly defining VE applications by using high-level conceptual models. We focused on the different steps of the VR-DeMo approach and explained how they are supported by two software development tools. Finally, using a case study, we demonstrated that the designer can model a VE application from a high-level away from any implementation details using the VR-DeMo approach. Using high-level models and notations, a city park design application. This case has shown that, compared to the traditional coding of VE applications, the VR-DeMo approach allows to more rapidly create VR applications with the additional benefit of being to more easily change parts of the virtual environment, the behaviour of the objects and the interaction with the end user.

7- Acknowledgements

The VR-DeMo project is directly funded by the IWT, a Flemish subsidy organization (IWT 030248).

The authors would like to thank Erwin Cuppens, Joan De Boeck, Tom De Weyer and Bram Pellens for their work in this project.

8- References

[1] Bierbaum A., Just C., Hartling P., Meinert K., Baker A. and Cruz-Neira C. VR Juggler: A Virtual Platform for Virtual Reality Application Development. In Proceedings of IEEE Virtual Reality Conference 2001, Yokohama - Japan, 2001

[2] Tanriverdi V. and Jacob R.J.K. VRID: A Design Model and Methodology for Developing Virtual Reality Interfaces. In Proceedings of ACM Symposium on Virtual Reality Software and Technology, Alberta - Canada, 2001

[3] Willans J.S., Harrison M.D. and Smith S.P. 'Implementing virtual environment object behavior from a specification' In User Guidance in Virtual Environments, Shaker Verlag, Aachen - Germany, pp. 87 – 97, 2000

[4] Fishwick P., Lee J., Park M. and Shim H. RUBE: a customized 2d and 3d modelling framework for simulation. In Proceedings of the 35th Conference on Winter Simulation: Driving Innovation, New Orleans -USA, 2003

[5] Kim G.J., Kang K.C., Kim H. and Lee J. Software Engineering of Virtual Worlds. In Proceedings of the ACM Symposium on Virtual Reality Software & Technology, Taipei - Taiwan, 1998

[6] Virtools Dev, http://www.virtools.com/, Accessed August 2, 2006

[7] Palanque P. and Bastide R. Petri net based design of user-driven interfaces using the interactive cooperative objects formalism. In Proceedings of Interactive Systems: Design, Specification, and Verification 1994, Carrara - Italy, 1994

[8] Ambler S. Object Primer, The Agile Model-Driven Development with UML 2.0. Cambridge University Press, 2004

[9] Dragicevic P. and Fekete J.D. Support for input adaptability in the ICON toolkit. In Proceedings of the 6th international conference on multimodal interfaces, State College - USA, 2004

[10] Navarre D., Philippe Palanque P., Bastide R., Schyn A., Winckler M., Nedel L. and Freitas C. A Formal Description of Multimodal Interaction Techniques for Immersive Virtual Reality Applications. In Proceedings of Tenth IFIP TC13 International Conference on Human-Computer Interaction, Rome – Italy, 2005

[11] Figueroa P., Green M. and Hoover H.J. InTml: A Description Language for VR Applications. In Proceedings of Seventh international conference on 3D Web technology, Tempe, USA, 2002

[12] Willans J. and Harrison M. A toolset supported approach for designing and testing virtual environment interaction techniques. In International Journal of Human-Computer Studies, 55(2): 145-165, 2001.

[13] Calvary G., Coutaz J., Thevenin D., Limbourg Q., Bouillon L. and Vanderdonckt J.. A Unifying Reference Framework for Multi-Target User Interfaces. In Interacting with Computers, 15(3): 289–308, 2003.

[14] Limbourg Q., Vanderdonckt J., Michotte B., Bouillon L., Florins M. and Trevisan D. UsiXML: A User Interface Description Language for Context-Sensitive User Interfaces. In Proceedings of the ACM AVI'2004 Workshop "Developing User Interfaces with XML: Advances on User Interface Description Languages", Gallipoli - Italy, 2004

[15] Gonzalez J.M., Vanderdonckt J. and Arteaga J.M. A Method for Developing 3D User Interfaces of Information Systems. In Proceedings of 6th Int. Conf. on Computer-Aided Design of User Interfaces CADUI'2006, Bucharest - Romania, 2006

[16] OWL, http://www.w3.org/2004/OWL/, Accessed August 24th, 2006

[17] Kleinermann F., De Troyer O., Mansouri H., Romero R., Pellens B. and Bille W. Designing Semantic Virtual Reality Applications, In Proceedings of the 2nd INTUITION International Workshop, Senlis - France, 2005

[18] Bille W., De Troyer O., Pellens B. and Kleinermann F. Conceptual Modelling of Articulated Bodies in Virtual Environments Virtual Environments. In Proceedings of the 11th International Conference on Virtual Systems and Multimedia, Gent - Belgium, 2005

[19] Pellens B., De Troyer O., Bille W., Kleinermann F. and Romero R. An Ontology-Driven Approach for Modelling Behavior in Virtual Environments. In Ontology Mining and Engineering and its Use for Virtual Reality workshop, Agia Napa - Cyprus, 2005

[20] Pellens B., Kleinermann F., De Troyer O. and Bille, W. Model-Based Design of Virtual Environment Behavior, In Proceedings of the 12th International Conference on Virtual Systems and Multimedia, Xi'an - China, 2006

[21] Paternò F. Model-Based Design and Evaluation of Interactive Applications, Springer, 2000

[22] Cuppens E., Raymaekers C. and Coninx K.. VRIXML : A User Interface Description Language for Virtual Environments. In Proceedings of Developing User Interfaces with XML: Advances on User Interface Description Languages, Gallipoli - Italy, 2004

[23] De Boeck J., Raymaekers C. and Coninx K. Comparing NiMMiT and Data-Driven Notations for Describing Multimodal Interaction. In Proceedings of 5th workshop on TAsk MOdels and DIAgrams for user interface design, Hasselt – Belgium, 2006

[24] Vanacken D., De Boeck J., Raymaekers C. and Coninx K. NiMMiT: a Notation for Modelling Multimodal Interaction Techniques. In Proceedings of International Conference on Computer Graphics Theory and Applications, Setúbal – Portugal, 2006

[25] Pierce J., Stearns B. and Pausch R. Voodoo Dolls: seamless interaction at multiple scales in virtual environments. In Proceedings of symposium on interactive 3D graphics, Atlanta – USA, 1999

[26] Taylor R.M, Hudson T.C., Seeger A., Weber W., Juliano J. and Helser, A.T. VRPN: A Device-Independent, Network-Transparent VR Peripheral System. In Proceedings of ACM Symposium on Virtual Reality Software & Technology 2001, Banff – Canada, 2001

[27] Nigay L. Coutaz J. A Generic Platform for Addressing the Multimodal Challenge. In Proceedings of ACM CHI'95 Conference on Human factors in Computing Systems, Denver – USA, 1995

[28] Walker M., Eaton N.J. and Eaton N. (2003) ‘Microsoft Office Visio 2003 Inside Out’, Microsoft Press, ISBN: 0735615160

[29] X3D, X3D International Standards, http://www.web3d.org/x3d/specifications/, Accessed May 31, 2006

[30] LUA Language, http://www.lua.org/, Accessed 24th August, 2006

[31] Salembier P and Smith J R (2001) MPEG-7 Multimedia Description Schemes. IEEE transactions on circuits and systems for video technology. 11(6): 748-759

[32] Luyten K., Clerckx T., Coninx K. and Vanderdonckt J. Derivation of a Dialog Model from a Task Model by Activity Chain Extraction. In Proceedings of Interactive Systems: Design, Specification, and Verification, 2003

[33] Poupyrev I, Billinghurst M., Weghorst S. and Ichikawa T. The Go-Go Interaction Technique: Non-linear Mapping for Direct Manipulation in VR. In Proceedings of the 196 ACM Symposium on User Interface Software and Technology (UIST), Seattle - USA, 1996

[34] Mine M. and Brooks F.P. Moving Objects in Space: Exploiting Proprioception in Virtual Environment Interaction. In Proceedings of the SIGGRAPH 1997 annual conference on Computer graphics, Los Angeles – USA, 1997

[35] Ware C. and Osborne S. Exploration and Virtual Camera Control in Virtual Three Dimentional Environments. In Computer Graphics, 24(2), 1990