Conceptual Modeling for Virtual Reality

Olga De Troyer, Frederic Kleinermann, Bram Pellens, and Wesley Bille

WISE Research Lab
Vrije Universiteit Brussel
Pleinlaan 2, B-1050 Brussel, Belgium

{olga.detroyer,frederic.kleinermann,bram.pellens, Wesley.Bille}@vub.ac.be

Abstract

This paper explores the opportunities and challenges for Conceptual Modeling in the domain of Virtual Reality (VR). VR applications are becoming more feasible due to better and faster hardware, and due to new technology and faster network connections they also start to appear on the Internet. However, the development of such applications is still a specialized, time-consuming and expensive process. By introducing a Conceptual Modeling phase into the development process of VR applications, a number of the obstacles preventing a quick spread of this type of applications can be removed. However, existing Conceptual Modeling techniques are too limited for modeling a VR application in an appropriate way. The paper will show how Conceptual Modeling can be done for VR and how this may make VR more accessible to non VR-specialists. Furthermore, the paper will explain how Conceptual Modeling embedded in a semantic framework can provide the basis for semantically rich VR application, which may be essential for its success in the future and its use in the context of the Semantic Web. The paper will also point to some open research problems.

Keywords: Virtual Reality, Conceptual Modeling, Semantics.

1 Introduction

Conceptual Modeling has been used with success in different domains such as Information Systems, Web Information Systems, User Interface Modeling, and Software Engineering. It has less been used in domains like 3D Modeling and Virtual Reality (VR). VR is a technology to simulate environments and create the effect of an interactive three-dimensional world in which objects have a sense of spatial and physical presence and can be manipulated by the user as such. VR has gained a lot of popularity during the last decennia due to e.g., games and applications such as Second Life (Second Life 2007). Although a lot of tools are available for developing VR applications, it is time-consuming, expensive, complex and specialized. One of the reasons is that the development of VR applications directly starts at the implementation level. The virtual world that needs to be created must be expressed in terms of low level VR building blocks, such as textures, shapes, sensors, interpolators, etc. This requires a considerable amount of background knowledge in VR. In addition, it makes the gap between the application domain and the level at which the virtual world needs to be specified very large, and this makes the translation from the concepts in the application domain into implementation concepts a very difficult issue.

Like for other domains, introducing a Conceptual Design phase in the development process of a VR application may help the VR community in several ways. As Conceptual Modeling will introduce a mechanism to abstract from implementation details, it will reduce the complexity of developing a VR application. In addition, if well done, such an abstraction layer can also hide the specific jargon used in VR and then no special VR knowledge will be needed for making the conceptual design. Therefore, also non-technical people (like the customer or the end-user) can be involved in the development and this will improve the communication between the developers and the other stakeholders. In addition, by involving the customer more closely in the design process of the VR application, earlier detection of design flaws is possible. All this could help in realising more VR applications in a shorter time.

However, Conceptual Modeling for VR poses a lot of challenges as in VR applications a number of aspects, not present in classical software or information systems, are very essential. For example, VR applications are 3D worlds composed of 2D and 3D objects and often deal with 3D complex objects, for which the way the parts are connected will influence the way the complex objects can behave. Furthermore, to realize dynamic and realistic worlds, objects may need complex (physical) behaviors. Therefore, new modeling concepts, not present in classical conceptual modeling languages such as ER, ORM and UML, are needed.

The paper will show how Conceptual Modeling can be realized for VR applications. Furthermore, the paper will explain how semantically based Conceptual Modeling can provide the basis for semantically rich VR applications. The rest of this paper is structured as follows. Section 2 will provide an introduction to VR. We will discuss the different components of a VR application as well as how VR applications are developed these days. In section 3, we will discuss the limitations of current conceptual modeling techniques with respect to the

modeling of VR. Then, section 4 will introduce a conceptual modeling approach developed for VR and section 5 will explain into more detail some of its modeling concepts. Section 6 will point out the benefits of using an approach that is grounded into ontologies. In section 7 we will discus related work. Section 8 points out some open research problems. Finally, section 9 concludes the paper.

2 Virtual Reality (VR)

There are many definitions of Virtual Reality (VR) (Vince 2004 and Burdea 2003). Usually, the restrictive approaches define VR as three-dimensional (3D), multi-sensorial, immersive, real time, and interactive simulations of a space that can be experienced by users via three-dimensional input and output devices. For the context of this research, VR is defined as a three-dimensional computer representation of a space in which users can move their viewpoints freely in real time. We therefore consider the following cases being VR: 3D multi-user chats (Active Worlds (ActiveWorld 2007), first person 3D videogames (Quake (Quake 2007) and Unreal tournament (Unreal 2007)) and 3D virtual spaces on the Web (such as those created with VRML (Hartman and Wernecke 1998), X3D (Walsh and Sevenier 2005)).

In this section we will first define the main components of a VR application and then we will review how these main components are being modeled today.

2.1 Main components of a VR application

A VR application is made of different components (Vince 2004), which can be summarized as:

1) **The scene and the objects.** The scene corresponds to the world in which the objects are located. It contains lights, viewpoints and cameras. Furthermore, it has also some properties that apply to all the objects being located inside the virtual world. For instance, gravity can be a property of the world that applies to all its objects. The objects have a visual representation with color and material properties. They have a size, a position and an orientation.

2) **Behaviors**. The objects may have behaviors. For instance, they can move, rotate, change size and so on.

3) **Interaction.** The user must be able to interact with the virtual world and its objects. For instance, a user can pick up some objects or he can drag an object. This may be achieved by means of a regular mouse and keyboard or through special hardware such as a 3D mouse or data gloves (Vince 2004).

4) **Communication**. Nowadays, more and more VR applications are also collaborative environments in which remote users can interact with each other. To achieve this, network communications is important. We will not elaborate on this aspect, as we will not consider it in this paper.

5) **Sound.** VR applications also involve sound. Some research has been done over the last ten years in order to simulate sound in VR application. In this paper, the modeling of the sound will also not be addressed.

2.2 Developing a VR application

The developing of the different components of a VR application is not an easy task and during the last fifteen years, a number of software tools have been created to ease the developer's task. These tools can be classified into authoring tools and software programming libraries.

Authoring tools. Authoring tools allow the developer to model the static scene (objects and the scene) at a level that is higher than the implementation level. Nevertheless, they assume that the developer has some knowledge on VR and some programming skills to program behaviors using scripting languages. The scripting languages used by these authoring tools can change from one authoring tool to another.

Software Programming Libraries. With programming libraries a complete VR application can be programmed from scratch. An example of such a programming library for VR is Java3D (Palmer 2001). To use such a library, good knowledge about programming and a good knowledge about VR and computer graphics are required. The code needs to be compiled and linked before it can be executed. It is also possible to use a player which can, at run-time, interpret a 3D format and build the VR application. VRML (Hartman and Wernecke 1998) and X3D (Walsh and Sevenier 2005) are 3D formats that can be interpreted by special player through a Web browser. Examples of such players are the Octaga player (Octaga 2007) and the Flux player (Flux 2007).

We will know review how the first three components mentioned in section 2.1 are developed nowadays and which authoring tool and software programming libraries are used for each of these components.

2.2.1 The Scene and the Objects

Nowadays, a number of authoring tools exist for modeling the objects and the scene without having to program them. Each of these tools comes with their own features and GUI. Therefore, as a developer, it is important to know what features these tools support and how they are used via the GUI. The developer also needs to know which file formats are supported. According to the player (or APIs) that will be used and the cost of these authoring tools, the developer can decide which authoring tools to use. The most popular authoring tools are 3D Studio Max (Murdock 2002), Maya (Maya 2006), MilkShape 3D (Milkshape3D 2006), various Modelers at PlanIt 3D (PlantID 2006), AC3D (AC3D 2006), and Blender (Blender 2006). If the developer is developing the VR application for a certain file format, he needs to pay attention to the file formats supported by the authoring tool. For instance in the case of the VRML/X3D file format, not all of the tools allow exporting the objects into this format. Furthermore, they do not always export them correctly when behaviors are embedded. Beside the authoring tools, software programming libraries can be used to model the scene. But for this, the developer needs to have programming

skill. Among the existing libraries, there is Performer (Performer 2007), Java3D (Palmer 2001), X3D toolkit written in C++ (X3D toolkit 2007) or Xj3D (Xj3D 2007) written on top of Java3D. To create the materials (texture and color) of the objects, the authoring tools can be used to add this to the object being created. However, it is sometimes not sufficient, as the developer may need to be able to modify them to suit his/her VR application.

2.2.2 Behavior

Developing the behaviors is usually the most difficult task. Often, behaviors are specified using scripting languages (Flanagan 2001 and Gutschmidt 2003) or programmed by means of traditional languages like Java or C/C++. Nevertheless, some authoring tools allow modeling the behavior directly using scripting languages. This means that the developer still needs to know these scripting languages. Furthermore to create realistic behaviors, physics engines need to be used. This also increases the complexity of a VR application.

2.2.3 Interaction

Interaction can happen in two different ways. The first way concerns the interaction between objects. When an interaction happens, the objects can then perform certain behaviors triggered by the interaction. For instance, after an object has collided with another object, its shape (or its color) could change. For this, it is important to have collision detection. As the supported collision detection algorithm is not always very accurate, the developer may need to extend it or find ways to improve it. The second way of interaction concerns the interaction between the end-user and the objects. The software programming libraries usually support programming user interactions.

2.2.4 Navigation

Nowadays, virtual worlds can be very large. Therefore, it is important to consider how the user can navigate inside the virtual world without being lost in the virtual world. There are different ways to deal with navigation (e.g., tour guides), but to realize them the developer usually needs to have good programming skills and VR knowledge.

2.2.5 Conclusion and Reflections

Although there are quite a number of tools to help a developer to build a VR application, until now, the general problem with these tools and formats is that they are made for VR specialists or at least for people having programming skills and background in computer graphics or VR. The problem is that the VR application one wants to create must be expressed in terms of (combinations of) low-level building blocks of the VR technology. In addition, there is also no well-accepted development method for VR. Most of the time, a VR-expert meets the customer (often the application domain expert), and tries to understand the customer's requirements and the domain for which the VR application is going to be built. After a number of discussions, some sketches are made and some scenarios are specified. Then, the VR-expert(s) start to implement. In other words, the requirements are almost directly translated into an implementation. This way of working may result into a number of problems due to the VR-expert being not familiar with the application domain. The VR-expert may lose a lot of time to get acquainted with the domain. Usually, the first implementation of the VR application often presents many shortcomings and usually does not match all the expectations and requirements of the customer. For this reason, several iterations are usually needed before the result reaches an acceptable level of satisfaction for the customer. Therefore, the development process is time consuming, complex and expensive. In fact, the process of building a VR application can be compared to the situation in the domain of databases in the early 70s when developing a database was also not an easy task and required technically skilled people. It was only after the introduction of the Relational Model as an abstraction layer on top of the physical database structures, and the introduction of Conceptual Modeling techniques (such as ER and NIAM), which closed the gap between the application domain and the implementation, that the domain of databases became more accessible to a wider audience.

Therefore, like for other domains, introducing a Conceptual Design phase in the development process of a VR application can help the VR community in several ways. As Conceptual Modeling will introduce a mechanism to abstract from implementation details, it will reduce the complexity of developing a VR application and it avoids that people need a lot of specific VR knowledge for such a Conceptual Design phase. Therefore, also non-technical people (like the customer or the end-user) can be involved and this will improve the communication between the developers and the other stakeholders. In addition, by involving the customer more closely in the design process of the VR application, earlier detection of design flaws is possible. And finally, if the conceptual models describing the VR system are powerful enough, it may be possible to generate the system (or at least large parts of it) automatically (cf. Model-Driven Architecture (MDA) of the Object Management Group (OMG 2007)).

3 Limitations of Existing Conceptual Modeling Techniques

Several general-purpose conceptual modeling languages exist. Well-know languages are UML (Fowler and Scott 1999), ER (Chen 1976) and ORM (Halpin 2001). ER and ORM were designed to facilitate database design. Their main purpose is to support the data modeling of the application domain and to conceal the more technical aspects associated with databases. UML is broader and provides a set of notations that facilitates the development of a complete software project. However, these existing Conceptual Modeling techniques are too limited for modeling a VR application in an appropriate way.

To a certain extend, ORM and ER could be used to model the static structure of a VR application (i.e. the scene and the objects), however, both are lacking modeling concepts in terms of expressiveness towards VR

modeling. For example, they do not have built-in modeling concepts for specifying the position and orientation of objects in the scene or for modeling connected objects using different types of connections. Although, it is possible to model these issues using the existing modeling primitives, this would be tedious. E.g., each time the modeler would need a particular connection he would have to model it explicitly, resulting in a lot of "redundancy" and waste of time. In addition, the models constructed in this way would not be powerful enough to use them for code generation because the necessary semantics for concepts like connections would be lacking. Furthermore, neither ORM nor ER provides support for modeling behavior.

UML is more expressive than standard ORM and ER since it offers a number of diagram types to model the system dynamics. In principle, UML can be used to model a VR application. For example, the static part (the scene and the objects) can be modeled using class diagrams; the behavioral part by means of state chart diagrams; and the interaction can be described using the sequence diagrams. But again, although UML can be used to model a VR application from a software engineering point of view, it is also lacking expressiveness for VR modeling. However, in contrast to ORM and ER, UML offers the concept of stereotype to add new modeling primitives. A stereotype is a new type of modeling element that extends the semantics of the meta model. Stereotypes must be based on existing types or classes in the meta model. So, stereotypes allow extending the vocabulary of modeling concepts of UML. Extensions made by means of stereotypes will appear as basic building blocks in UML. So actually, they can be seen as first class citizens of UML. In principle, we could use these stereotypes to define the modeling primitives lacking for VR. However, these stereotypes also have some disadvantages. As stated by Berner, Glinz, and Joos (1999), stereotypes increase the complexity of the language. In addition, it is not clear if the concept of stereotype will be powerful enough to define all the modeling concepts needed and use them for code generation. UML also contains much more core concepts than needed for our purpose. Furthermore, the use of UML would also force the use of its modeling paradigm. It is our opinion, that for certain aspects of VR, this would not be the best solution. If we want to make VR more accessible for non VR-experts and in particular for application domain experts, we should carefully watch the intuitiveness of the conceptual modeling approach taken. The modeling approach of UML is very close to the way software is implemented by means of OO programming languages.

As current modeling languages are too limited for our purpose, we either need to extend them or we should define a new one. We have opted for the last solution because this provides more freedom in the development of the modeling concepts. However, it is clear that a lot of the modeling concepts used are not new and inspired by concepts for other modeling approaches.

4 A Conceptual Modeling Approach for VR Development

To facilitate and shorten the development process of VR applications we propose to introduce an explicit conceptual design phase in the development life cycle of a VR application. During this conceptual design phase, conceptual specifications (so-called conceptual models) are created. Such a conceptual specification will be a high-level description of virtual world, the objects inside the virtual world, the relations that hold between these objects and how these objects behave and interact with each other and with the user. These conceptual specifications must be free from any implementation details and therefore the approach should offer a set of high-level modeling concepts (i.e. a modeling language) for building these conceptual specifications. In addition, we require that these modeling concepts are very intuitive, so that they can be used, or at least be understood, by different stakeholders. This means that the vocabulary used, should be familiar to most of its users. If we also opt for a model-driven approach, the expressive power of the different modeling concepts must be sufficient to allow using the resulting models as input for an automatic implementation phase.

VR-WISE (**V**irtual **R**eality – **W**ith **I**ntuitive **S**pecifications **E**nabled) (De Troyer et al. 2003 and Bille et al. 2004) is such a conceptual model-based approach for VR-application development. Figure 1 illustrates the VR-WISE development process. Since the gap between the conceptual level and the implementation level is too large to be bridged in one single step, the development process is divided into three main phases, the *conceptual specification phase*, the *mapping phase* and the *generation phase*. Note that the sequential order as showed in the figure only shows the main process flow. In practice, the process will be iterative. One can do part of the conceptual specification, part of the mapping, invoke the generation and then go back to the specification to complete or correct it until everything is specified and the result is satisfying. We will explain each phase briefly in the following sub sections.

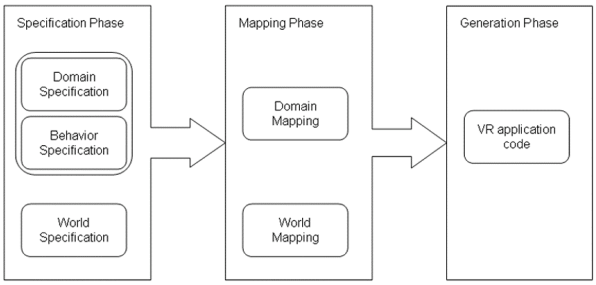

Figure 1: VR-WISE development process

4.1 The Conceptual Specification Phase

During the conceptual specification step the designer can specify the virtual world at a high-level using the intuitive modeling concepts offered by the VR-WISE approach. The specification consists of two levels since the approach follows to some degree the object-oriented paradigm.

The first level is the *domain specification*. The domain specification describes the concepts of the application domain needed for the virtual world (comparable to object types or classes in OO design methods). It also describes possible relations that may hold between these concepts. In the urban domain, the domain specification could contain concepts such as car, street, road sign, streetlight, building, and owner and relations such as "a building has an owner" and "a streetlight is located on a street". Concepts may have properties (attributes). Next to properties that may influence the visualization of the concepts (such as height, color, and material) also non-visual properties like the rent, and the function of building can be specified.

For VR applications behavior is an important feature. E.g., for an urban designers application it may be necessary to allow adding, removing and replacing road signs, rescaling and repositioning streets, and simulate the working of the streetlights by simulating some traffic. Therefore the conceptual specification phase also contains the behavior specifications.

The second level of the specification is the *world specification*. The world specification contains the conceptual description of the actual virtual world to be built. This specification is created by instantiating the concepts given in the domain specification. These instances actually represent the objects that will populate the virtual world. In the urban example, there can be multiple street-instances and multiple building-instances. Furthermore, behaviors specified at the domain level are assigned to objects and it is specified how these behaviors can be triggered (e.g., by means of a user interaction, a collision detection, or a time event).

In principle no or little knowledge about VR is needed to perform this phase.

Some of the modeling concepts devised to support this conceptual specification step are described in section 5.

4.2 The Mapping Phase

The purpose of the mapping step is to bridge the gap between the conceptual specifications and the implementation. Appealing visualizations and graphics are very important in the field of VR, therefore it is necessary to allow describing how the objects should be visualized in the virtual world. Similar as for the conceptual specification step, this is done at two levels. In the *domain mapping*, the designer specifies how the concepts from the domain specification should be visualized by means of VR implementation concepts or existing 3D models. For example, in a simple world a building could be mapped on a box indicating that it should be visualized as a box, but for a more demanding application a building could be mapped onto a 3D model of a building created by means of an authoring tool such as 3D Studio Max (Murdock 2002). The purpose of these mappings is to specify defaults for the visualization of the instances of the concepts in the virtual world. Although instances may be of the same concept, they may in some case require a different representation in the virtual world. Therefore the *world mapping* allows the designer to override these default mappings for specific instances. For example, you may want to have different representations for some (or all) of the building instances.

For this phase, the help of a VR-specialist may be needed, especially when 3D models need to be created or non-trivial mappings are needed.

Because the mapping phase is outside the scope of this paper, we will not elaborate on this step further on. More information can be found in (De Troyer et al. 2007).

4.3 The Generation Phase

During this step the actual source code for the virtual world is generated. This means that the conceptual specifications are converted into a working application by means of the mappings given during the mapping phase. In principle, different VR implementation languages can be supported. The current tool supporting VR-WISE allows generating X3D. Also this phase is outside the scope of the paper and will not be described further on. For more detail on this, we refer to (Pellens, 2007a).

5 Conceptual Modeling Concepts

In this section, we will describe a number of the modeling concepts that have been developed in the context of VR-WISE to model VR-applications. As common for conceptual languages, the modeling concepts also have a graphical notation, which we will also provide. First we describe modeling concepts for specifying the static structure of a virtual world, next we deal with the modeling of behavior.

5.1 Modeling the Static Structure

As already indicated, we distinguish between concepts and instances. A *concept* represents an object type from the application domain that is relevant for the VR-application. A concept can have a number of visual as well as non-visual *properties*, which can be given default values. A concept is graphical represented as a rectangle containing the name of the concept (see Figure 2a). The properties can be specified using the extended graphical notation (illustrated in Figure 2b).

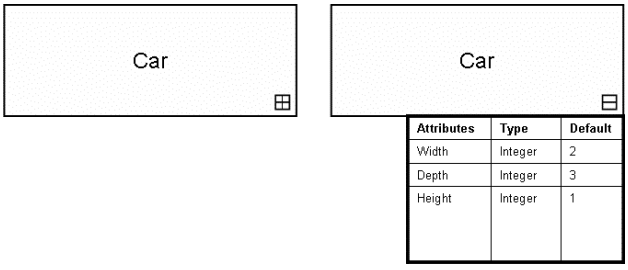

Figure 2: Example of a concept

The VR-objects in the actual virtual world are modeled as *instances* of concepts. In this way an instance inherits all the properties defined for the concept. Graphically, instances are represented as an ellipse containing the name of the concepts and the name of the instance separated by a colon (see Figure 3).

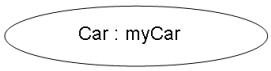

Figure 3: Example of an instance

Many of the modeling primitives that we will introduce are applicable to concepts as well as for instances. If this is the case the term *object* will be used.

In VR application, many objects are in fact assemblies of other objects like a human, which is an assembly of a head, 2 arms, 2 legs and a body. Therefore, we distinguish between *simple objects* and *complex objects*. In section 5.1.3, we will deal with issues related to the modeling of complex objects and explain some of the modeling primitives introduced for modeling them.

Another important aspect that needs to be modeled for a VR application is the scene. The scene contains the VR-objects. These objects have a position and an orientation in the scene (defined in a three-dimensional coordinate system). To model this we have introduce *spatial* and *orientation relations*. We explain these modeling concepts in section 5.1.1 and 5.1.2.

5.1.1 Spatial Relations

Although it is possible to specify the position of the instances in a scene by means of exact coordinates, we found it useful to provide a more intuitive way to do this. If you want to explain to somebody how your room looks like, you will not do this in term of coordinates. Instead you will say that "your bed is in front of the window, your desk is left of your bed, and at the right side of your bed there is a carpet on the floor". In such an explanation, spatial relations are used to describe the space. As spatial relations are also used in daily life, they provide a good intuitive way to specify a scene. Note that although the use of spatial relations may be less exact than coordinates, they are exact enough for a lot of applications. A *spatial relation* specifies the position of an object relative to some other object in terms of a direction and a distance. The following directions may be used: "left", "right", "front", "back", "top" and "bottom". These directions may be combined. However, not all combinations make sense. For example, the combined direction "left top" makes sense, but "left right" doesn't. Spatial relations can be used in the domain specification as well as in the world specification. In the domain specification, the spatial relations are used between concepts and specify default positions for the instances of a concept.

A spatial relation is graphically represented by a rounded rectangle (the general symbol for a relation) containing an icon indicating that the relation is a spatial relation. Below this icon the actual information for the spatial relation is specified: the direction and the distance. The graphical notation is illustrated in Figure 4, which expresses that the instance myCar (of the concept Car) is 3 meters in front of myHouse (which is an instance of House). Note that the spatial relation symbol is connected to the objects by means of an arrow. The direction of the arrow indicates that the instance myCar is in front of the instance myHouse and not vice versa. The direction of the arrow actually indicates the reading direction: myCar is in front of myHouse.

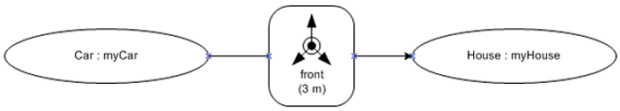

Figure 4: Example of a spatial relation

5.1.2 Orientation Relations

In VR, it is also necessary to indicate how objects are oriented. We can distinguish two types of orientations for an object: an *internal orientation* and an *external orientation*.

The internal orientation of an object is used to specify which side of the object is defined as the front, back, left, right, etc. The internal orientation is actually defined by a rotation of the local reference of the object around some of the axes of the default reference frame. This principle is illustrated in Figure 5. Figure 5(a) shows the default internal orientation of an object. In Figure 5(b), an internal orientation of 45 degrees counterclockwise around the front axis is illustrated. Note that the object itself is not rotated. Actually we only have redefined the left-right and top-bottom sides of the object.

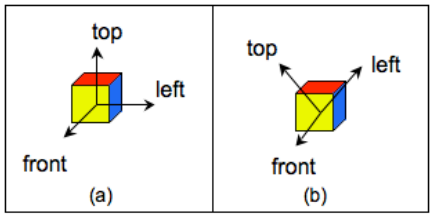

Figure 5: (a) Default internal orientation; (b) Non-default internal orientation

The external orientation of an object is used to rotate the object itself. This means that an object will be rotated around some of the axes of its reference frame and this will be visible in the virtual world. The external orientation is illustrated in Figure 6. Figure 6(a) shows the default orientation of an object while Figure 6(b) shows an external orientation with a rotation of 45 degrees counterclockwise around the front axis. As we can see, the concept itself has now been rotated.

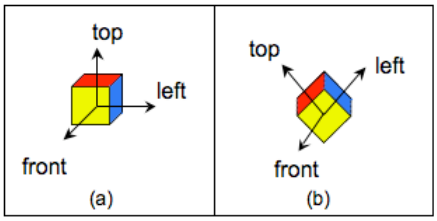

Figure 6: (a) Default orientation; (b) Non-default orientation

The internal orientation of a concept or instance can be specified by means of properties. To specify an external orientation different from the default one, we use *orientation relations*. There are two types of orientation relations. The first type, *orientation by side relation,* is

used to specify the orientation of an object relative to another object. It specifies which side of an object is oriented towards which side of another object. E.g., in Figure 7, it is specified that the instance myCar is oriented with its right side towards the front side of the instance myHouse. Also here, the direction of the arrow is important and indicates the reading direction.

Figure 7: Example of an orientation-by-side relation

The second type of orientation relation is the *orientation by angle relation*. This relation can be used to specify that an object is rotated around some axis of its local reference frame over a certain angle. Figure 8 illustrates the graphical notation for this modeling concept. It states that the external orientation of instance myCar is given by means of a rotation of 45 degrees around its top-to-bottom axis.

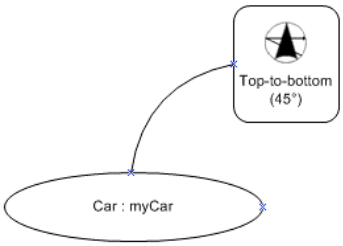

Figure 8: Example of an orientation-by-angle relation

5.1.3 Modeling Complex Objects

So far, we have seen how to specify simple concepts and instances and how to specify their position and their orientation. However, very often it is necessary to reflect in the virtual world that the VR-object is an assembly. Usually, all components of such an assembly should keep their own identity and it should be possible to manipulate them or let them behave individually as far as this should be allowed. E.g., a human avatar in a virtual world should be able to move his arm in the same way that the arm is limited to move for a human being. To model this, we use *complex objects*. Complex objects are defined using simple and/or other complex objects. They are composed by defining a connecting between two or more simple and/or complex objects. The connected objects are called *components*. In the virtual world, all components will keep their own identity and can be manipulated individually within the limits imposed by the connection. In VR, in general, different types of connections are possible. The type of connection used, has an impact on the possible motion of the components with respect to each other. We explain this in more detail. Normally an object has six degrees of freedom, three translational degrees of freedom and three rotational degrees of freedom. The translational degrees of freedom are translations along the three axes of the coordinate system while the three rotational degrees of freedom are the rotations around these three axes. Different types of connections will restrict the degrees of freedom in different ways. Therefore it is important to be able to model different types of connections. This is done by means of *connection relations*. We will present the *connection point relation*, the *connection axis relation*, and the *connection surface relation*.

For other modeling concepts developed to deal with complex objects (such as e.g., position and orientation of complex objects, instantiation of complex concepts, *roles,* and *connectionless complex objects*), we refer to (Bille 2007).

Connection Point Relation

The connection point relation allows modeling that two components of a complex object are connected to each other over a center of motion. In the real world we can find many examples of physical objects connected over a center of motion like the connection of the arm to the torso in a human body. A center of motion means that in the VR representation of both connected components there is somewhere a point that needs to fall together during the complete lifetime of the connection. We call this point the *connection point*. Connecting two objects over a center of motion removes all three translational degrees of freedom of the objects with respect to each other.

To specify a connection point relation between two objects we have to specify the connection point on each object. The position of a connection point is specified relative to the position of the object (which is given by the position of the middle point). This can be done using the spatial relations introduced in section 5.1.1. For example, the designer can specify that the connection point lies 3 centimeters left of the positioning point. Also here directions may be combined when they make sense.

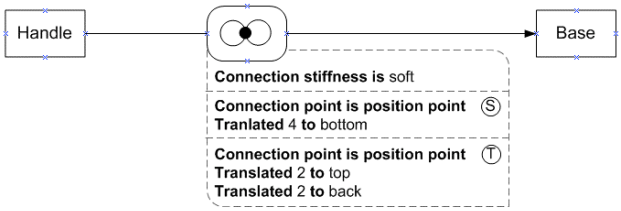

Figure 9: Example of a connection point relation

In Figure 9, the graphical notation for a connection point relation is illustrated. It specifies that a Handle is connected to a Base by means of a connection point relation. The Handle is called the *source* of the connection and the Base is *target* (indicated by the direction of the arrow). The connection points for the source (S) and the target (T) are specified using a simple script language in the extended version of the notation. In this example, the connection point of the Handle is 4 units from the positioning point of the Handle towards the bottom, and the connection point of the Base is 2 units backwards and 2 units towards the top from the positioning point of the base. In addition, the scripting language allows specifying a number of properties for the connection, such as the *stiffness*. The properties are specified by means of intuitive terms rather than by exact

values. E.g., possible values for the stiffness are “soft”, “medium”, and “hard”. In the example, the value “soft” has been used.

Connection Axis Relation

A second way to connect two objects is over an axis of motion. Again a lot of examples of this connection type can be found in the real world. Some examples are a wheel that turns around a certain axis; a door connected to a wall, which opens around a certain axis; the slider of an old-fashioned typing machine that moves along a certain axis. Actually, a connection over an axis of motion means that the displacements of the connected objects with respect to each other, is restricted to the movement along or around this axis. The axis of motion is called the *connection axis*. A connection by means of a connection axis removes four degrees of freedom leaving only one translational degree and one rotational degree of freedom.

To specify a connection axis relation between two objects we have to specify a connection axis for each object. These two axes need to fall together during the complete lifetime of the connection. Such an axis can be defined as the intersection between two planes. To facilitate this, through each object three planes are pre-defined. These are the horizontal plane (defined by the front-to-back and the left-to-right axes, see Figure 10(a)), the vertical plane (defined by the front-to-back and the top-to-bottom axes, see Figure 10(b)), and the perpendicular plane (defined by the left-to-right and the top-to-bottom axes, see Figure 10(c)). A connection axis is defined as the intersection between two of these planes. The three predefined planes can also be translated or rotated which allows more possibilities to define an axis.

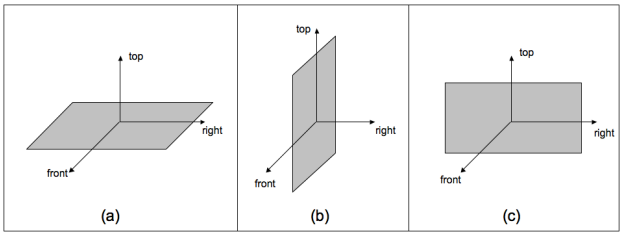

Figure 10: (a) The horizontal plane; (b) The vertical plane; (c) The perpendicular plane

Next to defining the connection axes, it is also necessary to give the initial positions of both components. For this we use *translation points*. A translation point is defined as the orthogonal projection of the middle point (or position) of the component onto its connection axis. By default, the components will be positioned in such a way that the connection axis falls together as well as the translation points. Note that the designer can also specify that the objects should be first translated along the connection axis (details are omitted here; they can be found in (Bille 2007)).

Figure 11 shows an example of two concepts, a Door and a DoorPost, connected to each other by means of a connection axis. Similar as for the connection point relation, there is a source and a target. Also for this relation, the details are given in the extended version of the graphical notation by means of the scripting language. The connection axis for the Door, the source, is specified by a translation of the vertical plane over half of the width of the Door (width has been defined as a property of Door) to the right and the perpendicular plane over half of the depth of the Door (depth being also a property of Door) to the front. The connection axis for the DoorPost, the target, is specified by a translation of the vertical plane over half of the width of the DoorPost (property) to the left and the perpendicular plane over half of the depth of the DoorPost (also a property of DoorPost) to the front. The translation points are not mentioned, which means that the defaults should be used. The stiffness attribute of the connection axis relation is set to “medium”. The specification of the connection axis is visually illustrated in Figure 12.

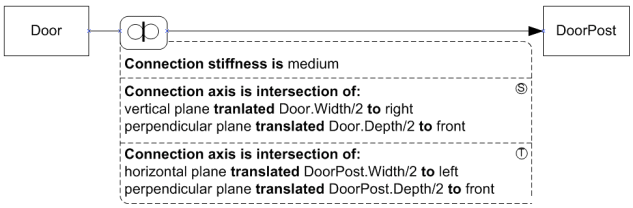

Figure 11: Example of a connection axis relation

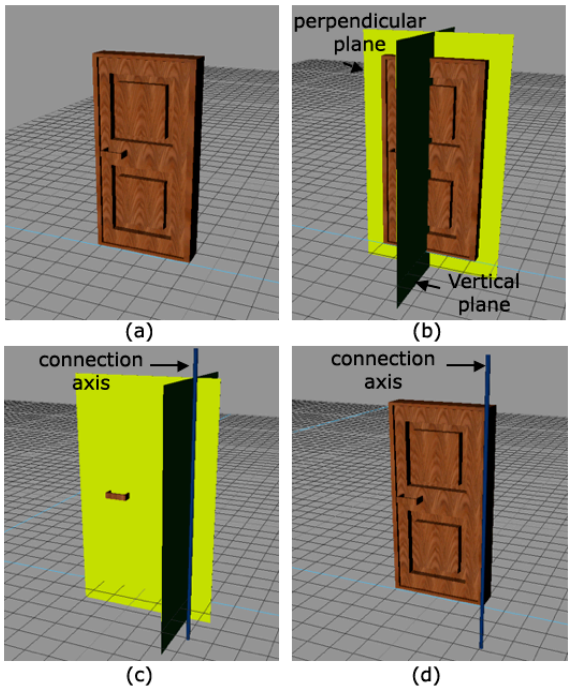

Figure 12: Illustration of a connection axis relation

Connection Surface Relation

A third way to connect two objects to each other is over a surface of motion. A real world example of this type of connection is a boat on a water surface. The boat should be able to float on the water surface. However, its bottom surface should stay inside the water surface. A surface of motion means that there is a surface along which the connected objects may move. This connection removes 3 degrees of freedom. The degrees of freedom possible in this case are the two translational degrees of freedom in

the directions of the surface and one rotational degree around the axis perpendicular to the surface. This is illustrated in Figure 13. The surface of motion is called the *connection surface*.

This kind of connections can be specified by means of the connection surface relation. To specify a connection surface relation, a connection surface on both components should be specified. The connection surfaces of both objects will need to fall together during the complete lifetime of the connection. For specifying these connection surfaces, the same pre-defined planes as mentioned for the connection axis relation, namely the horizontal plane, the vertical plane and the perpendicular plane, are used. However, now the designer should select only one of these planes. This plane can be translated and rotated to arrive at the desired connection surface. Similar as for the connection axis relation, it is also necessary to specify the initial position of both components along the connection surface by means of translation points. By default, the translation point of a component is the orthogonal projection of the middle point (position) of the object on its connection surface. The components will be connected in such a way that the connection surfaces as well as the translation points fall together. Also for a connection surface relation, the default translation points can be changed by means of translations (see again (Bille 2007)).

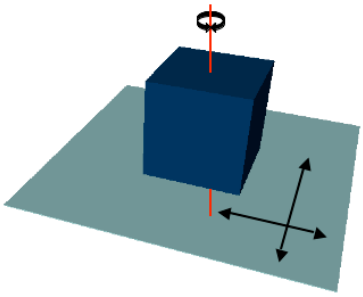

Figure 13: Degrees of freedom for the connection surface relation

Figure 14 illustrates the use of a connection surface relation to model the example of the boat on a water surface. The connection surface for the Boat object is defined as the horizontal plane translated 3 units towards the bottom of the Boat; the connection surface for the WaterSurface is also the horizontal plane. As the Water Surface is a plane itself, there is no need to translate this horizontal plane. Figure 15 shows a possible outcome of the specification given in Figure 14. This connection allows a Boat-instance to move freely on the Water Surface (represented as a blue plane).

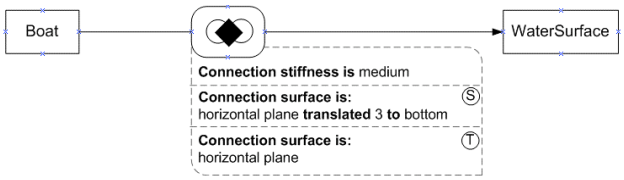

Figure 14: Example of a connection surface relation

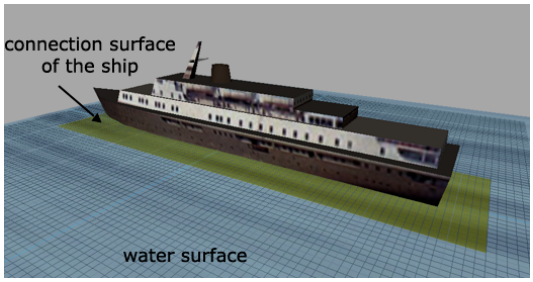

Figure 15: illustration of a connection surface relation

5.1.4 Constraints

Next to specifying the objects of the scene, their position and orientation in the scene, and the way complex objects are composed, it may be necessary to further constraint the specifications. This is done by means of so-called *constraints*.

As explained in section 5.1.3, a connection relation already imposes an implicit constraint on how the components can move. However, this may not be sufficient for realistic worlds. For example, consider again the example of the door and the doorpost. By using the connection axis relation, their motion is constrained to one translational degree and one rotational degree of freedom, but in reality a door cannot move along the doorpost and the angle over which the door can rotate is limited. To further restrict the movement, constraints are used. So far, a number of declarative types of constraints have been defined. They are based on metaphors to make it easy for non VR-skilled people to understand them. Some examples are the *hinge constraint*, the *slider constraint* and the *joystick constraint*. The hinge constraint can be defined on top of connection axis relation to restrict the motion to a rotation around the connection axis. Furthermore, the rotation can be limited by indicating how much the components may rotate around the connection axis in the clockwise as well as in the counterclockwise direction. Figure 16 illustrates a hinge constraint for the door example. The constraint specifies that from the initial position (the closed door), the Door can be pulled 90 degrees but cannot be pushed. Note that the metaphor is also used in the graphical notation.

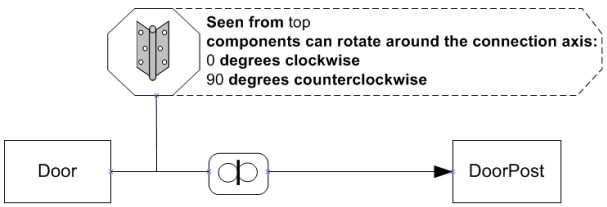

Figure 16: Example of a hinge constraint

The slider constraint is also specified on top of a connection axis constraint and restricts the motion to a move along the connection axis. It is also possible to indicate limits for this movement.

The joystick constraint restricts the motion of two components connected by means of a connection point relation to a rotation around two perpendicular axes through the connection point. A joystick constraint can also have limits indicating how much the components may rotate around the axes in the clockwise and in the counterclockwise direction.

So far, we have only considered constraints on objects that are physically connected. However it may also be necessary to constrain the motion of objects that are not physically connected. For example when simulating a magnetic field between two objects, the way the objects can move should be restricted, or we may want to enforce that a coffee cup can only be placed on a saucer. Some of these situations may occur quite often in virtual worlds. Therefore, a number of declarative constraints have been defined to cover these situations. Examples are: the *fixed relative position constraint*, the *fixed relative orientation constraint* and the *positioning constraint*. The fixed relative position constraint states that the initial relative position between two objects should be maintained during the complete lifetime of the objects, while for the fixed relative orientation constraint the initial relative orientation of two objects should be maintained. The positioning constraint allows restricting the positioning of objects by defining an anchor and binding areas for objects. More information on these constraints can be found in (Bille 2007).

5.2 Modeling Behavior

As mentioned earlier, in current VR approaches behavior is often specified directly using some scripting language. For our conceptual modeling approach, we also have taken a model-driven approach, meaning that high-level models are used to describe the behaviors and these models should be detailed enough to enable the generation of code. This means that also for describing behavior a number of high-level and intuitive modeling concepts are needed.

Most model-based approaches encountered in the related work (see section 7) use state-machines as the underlying model for describing behavior of objects. The descriptions are then based on the different states the object can be in during their lifetime. Our behavior modeling approach uses a different approach, which we call *action-oriented* because the approach is based on the different actions that an object may undertake throughout the lifetime of the application rather than on the states an object can be in.

A traditional animation process uses transformation operations for modifying the numerical data describing the objects in space. These transformation operations are represented by matrices or quaternions. Although modeling tools allow manipulating these transformations through more friendly user-interfaces, it still requires background knowledge in mathematics to correctly create such transformations. Our approach tries to provide a higher level of abstraction by uses more intuitive actions instead of transformations. When an animation is specified by means of a number of transformations, then these transformations need to be set correctly throughout time. These transformations are usually not only sequential but are often related to each other in a more complex way. One of the most difficult aspects for creating compelling behaviors is this time setting. Therefore, we have provided a number of modeling concepts that allow intuitive timing of the actions.

Furthermore, to reduce the complexity and to enhance reusability, the modeling of behavior is divided into two separate steps: the *Behavior Definition* and the *Behavior Invocation*.

The first step, the behavior definition, allows defining different behaviors. It should be noted that the actual specification of the behaviors is separated from the specification of the concepts and the instances that will have the behavior, as well as independent of how the behavior will be triggered. This improves reusability of behaviors and enhances flexibility as the same behavior definition can be used for different objects and/or can be triggered in different ways (e.g., by different user interactions or by collision with other objects). The binding of behavior to objects is done in the second step, the behavior invocation. Furthermore, in this second step it is also specified how the behaviors assigned to objects may be invoked, i.e. the events that may trigger them. We will now discuss the most important modeling concepts for each of these steps.

5.2.1 Behavior Definition

To separate the definition of the behaviors of an object from the definition of the static properties of the object, the concept of *actor* is used. An actor is used as a placeholder for an object when specifying behavior. It is also used to specify the minimal set of static properties that an object needs to have for the behavior. Later on, in the Behavior Invocation, a behavior can be assigned to an object when it has at least this minimal set of properties.

We distinguish between *primitive behaviors* and *composite behaviors*. We first discuss primitive behaviors; next composite behaviors. We mainly focus on the available modeling concepts. Note that most modeling concepts can be complemented with scripts to specify more details or to model more advanced behaviors. However, this script language is out of the scope of the paper. We refer to (Pellens 2007a) for this.

Primitive Behaviors

To express primitive behavior, a number of modeling concepts, called *actions*, representing these behaviors have been defined. They represent behavior that perform changes at the object level, such as *Move* (to change the position of an object), *Roll* (to specify a rotation of an object around its top-to-bottom axis), *Turn* (to express a rotation of an object around either its left-to-right axis and/or its front-to-back axis), *Resize* (to resize an object), *Position* (to specify a specific position for an object), *Orientate* (to specify a specific orientation for an object), and *Transform* (to specify a change in the appearance of an object (at runtime)). An example is given is given in Figure 17. The graphical representation of a behavior is a

box. The box is connected to at least one actor (for which the behavior is defined), but other actors way also be attached to the behavior if needed (see later on). An icon represents the type of behavior; in the example we see a an actor "Bus" has a move behavior with a combined direction, forward and left, over a distance of 100 meters; note that the direction of the move and the distance are also specified. The direction can have one of the values: "left", "right", "forward", "backward", "up", and "down". It is also possible to combine directions, e.g., "forward-left". Additional parameters are possible, e.g., the speed of the movement. For an elaborated discussion on these modeling concepts see (Pellens et al. 2007b or Pellens 2007a).

Figure 17: Example of a primitive behavior

There are also modeling concepts for behaviors that have an influence on the structure of the overall scene, such as *Construct* (to specify the creation of a new object at runtime), *Destruct* (to specify that an object should be removed from the scene), *Ungrouping* (to assemble objects into a complex object at runtime), *Grouping* (to disconnect the components of a complex object at runtime), *Disperse* (to specify that at runtime an object can be broken into —and replaced by— a number of new objects), *Combine* (to specify that at runtime some object can be combined into —and replaced by— a single new object). An example is given in Figure 18. A disperse behavior, BreakShelve, is specified. It specifies the behavior where a Shelve (the actor for which the behavior is defined) will be broken resulting in two new objects, namely a ShelvePiece and a RightSupport (modeled as output actors). The ShelvePiece will be positioned 1 cm left-of the RightSupport (expressed by a spatial relation in the middle part of the box). More details on these modeling concepts can be found in (Pellens et al. 2006 or Pellens 2007a).

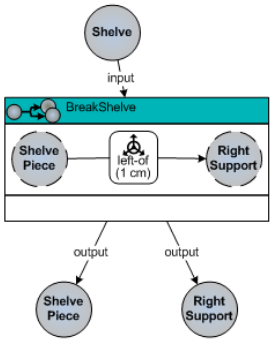

Figure 18: Example of a disperse behavior

Composite Behaviors

Different behaviors can be combined to form a so-called composite behavior. It allows defining more complex behaviors and also provides an abstraction mechanism. The concept of *operator* is used to compose behaviors. Four different kinds of operators are introduced, the *temporal operators*, the *lifetime operators*, the *conditional operator* and the *influential operator*.

Temporal operators provide a more intuitive way for specifying time-dependencies between behaviors than the typical key frame animation methods found in most VR modeling tools. They allow synchronizing behaviors. The temporal operators proposed are based on the binary temporal relations defined in (Allen 1991). However, some adaptations were needed in order to completely specify temporal relationships between behaviors. The operators supported in our approach are *before* (inverse *after*), *meets* (inverse *met-by*), *overlaps* (inverse *overlapped-by*), *during* (inverse *contains*), *starts* (inverse *started-by*), *ends* (inverse *ended-by*), and *equals*. In the graphical notation, an operator is drawn as a rectangle with rounded corners containing an icon that indicates the type of operator. The operator is connected to the behaviors involved by single solid lines. The arrow indicates the reading direction. In Figure 19, an example of a temporal operator is given. It specifies that a moving action of 5 meters in the forward direction needs to happen 5 seconds before a turning action of 90 degrees to the right.

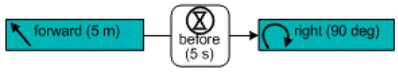

Figure 19: Example of a temporal operator

Sometimes, it must be possible to prohibit some behavior, or to put a behavior on hold and resume it afterwards. Therefore lifetime operators allow the designer to describe that one behavior controls the lifetime of other behavior(s). The following lifetime operators are supported: *enable*, *disable*, *suspend*, and *resume*. Figure 20 specifies that the execution of the UnlockDoor behavior will enable the OpenDoor behavior (which has been disabled in some way).

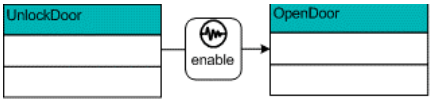

Figure 20: Example of a lifetime operator

Furthermore, the conditional operator can be used to control the execution of behaviours by means of conditions. The influential operator can be used to indicate inter-relationships of the behaviors. It allows specifying how the behavior of one object can influence the behavior of another object. The use of the influential operator is very useful when modeling mechanical devices e.g., gears, belts and pulleys. For example, in a rack-and-pinion gear (which converts a rotation into a linear motion), the pinion is rotating and this rotation engages the movement of the rack respectively according to a given ratio (e.g., x = 3/4 y).

5.2.2 Behavior Invocation

As explained before, the definition of behaviors is independent from the specification of the actual objects in the scene. In the Behavior Invocation diagrams, behaviors

are assigned to objects. Furthermore, they are also used to denote the events that may trigger the behaviors of the particular objects.

Behaviors are attached to an object by associating the object with the actor(s) for which these behaviors were defined. To specify when the behaviors can be invoked, the concept of *event* is used. We distinguish between *context-events*, *time-events*, *user-events*, and *object-events*.

The *context-events* enable the designer to specify the context (or situation) in which a behavior of an object needs to be invoked e.g., when the temperature goes beyond 25 degrees Celsius. A context is defined as a condition on some entities. Entities are objects, users, or the virtual world itself, considered to be relevant for the behavior in question.

A *time-event* allows the designer to specify the moment in time that the behavior needs to be triggered. This can be a relative time representing the time that has to pass counted from the start of the simulation; an absolute time; or a time schedule given by a duration and an optional from and to clause, for example: "1min FROM 13:00 TO 14:00" to indicate "every minute between 1 and 2 PM".

Using a *user-event*, the designer can specify that the behavior for an object needs to be triggered when some user interaction occurs. The following user-events are supported: *OnSelect* (when the user selects the object, i.e. it is clicked with the mouse or selected through another selection technique), *OnTouch* (when the user has the mouse or any other pointing device over the object), *OnVisible* (when the user can see the object), *OnProxy(p)* (when the user has entered a particular perimeter p around the object), and *OnKeyPress(k,m)* (when a particular key-combination, given by a key k and a mask m, is pressed on the keyboard). Next to these predefined user actions, also custom-made actions can be defined which allow behaviors to be triggered as a reaction on more complicated user interaction techniques (e.g., using menus, dialogs).

The last type of event, the *object-event* represents the event that is fired when two (or more) objects in the virtual world interact with each other. Two types of object events are distinguished. The *collision-event* allows reacting to the situation where an object encounters an obstacle in the form of another object, i.e. when a collision between two objects occurs. The *constraint-event* allows reacting when the limit of a particular constraint has been reached, or when a constraint has been violated.

An example of a Behavior Invocation diagram is given in. It specifies that the BusManoeuvre behavior is attached to the object bus1 and should be triggered by means of pressing a key.

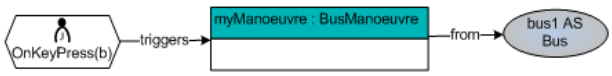

Figure 21: Example of a behavior invocation diagram

More details about behavior invocation can be found in (Pellens et al. 2007b or Pellens 2007a).

6 Towards Semantic Virtual Worlds

Although VR-applications are becoming visually appealing, they often lack any kind of semantics, i.e. extra, non-visual information about the virtual world and its objects (Martinez and Delgado Mata 2006). Usually, the information associated with a virtual world is limited to low-level information such as the type of geometry, the size and material. Furthermore, if any information is added to a virtual world, then this is often done after the virtual world has been created. This again increases the development time and cost of a VR application.

The information that is added to some media to enrich it is nowadays called semantic annotations. Semantic annotations are especially important in the context of the Semantic Web because they make the content of the Web machine-processable and enable computers and people to work in cooperation. Similar as for Websites, semantic annotations can be added to virtual worlds and to their objects. This is not only useful for making the content machine-processable, in the context of VR, semantic annotations are also very important to increase the usability of the virtual world and/or to adapt the virtual world to a particular task or user (as discussed by Kleinermann et al. (2007)). In particular, semantic annotations are very important for application domains where providing information is a major concern (e.g., Virtual Museums).

By the introduction of a conceptual modeling phase in the development process of a virtual world, it is possible to have semantic annotations automatically included in the virtual world. In VR-WISE, we have achieved this by using domain ontologies during the conceptual specification phase (Kleinermann et al. 2005 and De Troyer et al. 2007). A *domain ontology* is used to capture the concepts needed from the domain under consideration. As discussed in section 5.1, next to the visual properties needed to be able to generate the virtual world, the designer can add extra (domain-oriented) properties to concepts, instances, and behaviors. These are captured in the domain ontology. During the generation phase they are used to automatically generate semantic annotations or incorporate semantics directly into the virtual world. This semantic information may enhance the usability of the VR application. For instance, a search engine can exploit the semantic information by allowing more powerful and domain-oriented queries How this can be achieved is described in (Kleinermann et al. 2005 and De Troyer et al. 2007). Also inside the virtual world domain specific semantic information can be provided to the end-user (e.g., when clicking on an object or when approaching the object).

7 Related Work

The lack of high-level design methodologies for VR has also been addressed in (Tanriverdi and Jacob 2001) with the presentation of VRID (Virtual Reality Interface Design). In their paper, four key components when

developing VR interfaces are identified: object graphics, object behaviors, 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 developer to split the design into different steps and then refine them, it still does not allow the developer to express the design using domain terminology and relations. The low-level phase forces the designer to deal with low-level issues, which can be difficult and too complex for non-experienced designers.

The Virtual Environment Development Structure (VEDS) described in (Wilson 2002) is a user-centered approach for specifying, developing and evaluating VR applications. The main aim of this approach is to guide the designer in its design decisions in such a way that usability, likeability and acceptability are improved. This will eventually lead to a more widespread use of VR. With VEDS, the domain for which the VR application is developed is really integrated into the design stage. VEDS has a conceptual phase before the real development of the VR application. During that phase, the actual VR application is specified at a high level making balanced decisions on the goals that were set up at the beginning of the process. This specification is used by the developers to build the VR application. Furthermore, there is also a sort of iterative loop in which the design of the VR application is refined step by step until it meets the customer's expectations. Nevertheless, the domain expert is not very much involved into the actual design of the VR application and is solicited only at the beginning of the design phase.

The Concurrent and LEvel by Level Development of VR systems (CLEVR) approach looks at the design problem from a software engineering point of view and applies current techniques from this field to VR design. The CLEVR is the successor of the ADASAL/PROTO approach (Kim et al. 1998). The authors in (Kang et al. 1998) see a virtual world as a combination of three inter-related aspects: form, function and behavior. Although the approach provides a way to design VE applications, it is based on the assumption that the designer understands the UML notation and has knowledge about Object-Oriented (OO) design. It is very much based on classical software engineering principles. A more detailed description of the approach together with examples can be found in (Seo 2002).

The Ossa system is an approach to conceptually model of VR systems (Southey and Linders 2001). Ossa provides a modeling environment that allows building strong underlying conceptual models, as a sort of skeleton for the VR application. These models are a combination of conceptual graphs and production systems. The conceptual graphs are used for representing the knowledge of the world that is about to be designed. The production systems approach is taken to capture the dynamics of the application. A more detailed description can be found in (Southey 1998). The disadvantage of the Ossa system is the large complexity it brings since it is not using a normal procedural approach for specifying the dynamics. A rule-based approach is used resulting in more complicated execution patterns. Besides this, also the fact that the rules need to be described in a kind of logic programming style makes that they are probably not usable for non-skilled persons.

The lack of a proper design methodology is also acknowledged in the research performed in the context of the interactive 4D (i4D) framework (Geiger et al. 2001). I4D is a framework for the structured design of all kinds of interactive and animated media. The approach not only targets the domain of Virtual (and Augmented) Reality but also the domains of 3D graphics and multimedia. The i4D design approach aims to express the conceptual models in terms of concepts that are familiar to all the stakeholders of the application. In i4D, an actor-based metaphor is used. This forces to describe a VR application using a dedicated terminology, namely that of role-plays: the actors act like particular roles that are specified by the designer. Other domain knowledge cannot be used. Furthermore, most of the issues eventually need to be programmed in their framework, which only provides a thin abstraction layer on top of the currently existing graphics libraries.

In (Willans et al. 2001), the authors have developed software that separates the process of designing interaction techniques from the process of building a specific virtual world, making it easier for developers to design realistic interaction techniques and try them out on users. However, the way behaviors are being designed is still very much an engineering way and therefore, not intuitive for a non-engineer person.

The Rube methodology proposed by Fishwick (Fishwick 2000) 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.

A commercial development environment that has similar goals is Virtools Dev (Virtools Dev 2007). Virtools is not intended to be a fully functional modeling environment. It only has some basic support to compose the virtual scene. Virtools also allows the designer to define behavior for objects 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 behavior (and interaction), which shows the execution flow, together with additional data-flow. We consider the approach taken by Virtools still as a low-level.

Furthermore, several models and description languages exist, which can be used to define user interaction. Examples are Petri nets (Palanque and Bastide 1994), UML (Ambler 2004) and ICon (Dragicevic and Fekete 2004). Despite their focus on interaction, these models are very generic and are often cumbersome to use for describing interaction, particularly in VR applications. Other models, such as ICO (Navarre et al. 2005) and InTml (Figueroa 2002), have been developed with interaction in virtual world in mind. These models have the drawback that they are not easy to apply in a cognitive modeling approach, where the specified models have to be interpreted at runtime by the application. The

Marigold toolset (Willans 2001) is an 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 (Calvary et al 2003), and the UsiXML process for defining context-sensitive user interfaces (Limbourg et al. 2004), a method has been designed for creating 3D user interfaces (Gonzales et al 2006). However, the method needs further experimental validation.

8 Further Work and Open Research Problems

The modeling primitives currently available in VR-WISE are far from complete. We mention here some of the limitations of the modeling concepts presented, as well some missing concepts.

One limitation concerning the modeling of connections is that it is not yet possible to define a connection between two components that is a combination of connections, or to combine constraints on connections. This is needed to allow for more powerful connections such as e.g., a car wheel. For this we need actually a combination of two hinge constraints. The problem however is that the motion allowed by one hinge constraint may be in contradiction with the motion allowed by the second hinge constraint, and therefore a simple combination is not sufficient.

Next, we also need modeling concepts for specifying so-called contact joints. This type of joints does not really describe a connection but rather a contact between two objects like the gearwheels of a watch that need to roll against each other.

Also the behavior modeling has some limitations. We currently only consider event-driven behavior, where the behavior is triggered by an event and is then executed independently of this event. However, in many cases, interaction is much more intertwined with the behavior. This is called interaction-controlled behavior; during the complete duration of the behavior the user (interaction) is having control over the object.

Furthermore, we did not consider the modeling of interaction in this paper. This has been investigated by our partner in the VR-DeMo project, being the project is which part of this research has been conducted. We refer to (Coninx et al. 2006) for a description of VR-DeMo and to (Vanacken et al. 2006) for the modeling of interaction.

Other aspects of virtual worlds that are not yet covered are the modeling of sound and communication, the modeling of some properties of the world itself like cameras, viewpoints, light sources, and shadows. Also the use of avatars is an important issue in VR applications that have not been considered so far.

9 Conclusions

In this paper, we have described why conceptual modeling can be important for the field of Virtual Reality. We also explained the shortcomings of current conceptual modeling languages with respect to VR. Next, we have presented a conceptual modeling approach for VR and discussed some of its modeling concepts. We have concluded the paper by pointing to some of the limitations of the approach and by identifying open research problems.

10 Acknowledgements

This work was carried out in the context of the VR-DeMo project, funded by IWT (the Institute for the Promotion of Innovation by Science and Technology in Flanders). It was also partially funded by FWO (Fund of Scientific Research – Flanders).

11 References

Allen, J.F. (1991): Time and time again: The many ways to represent time. *International Journal of Intelligent Systems*, 6(4):341–355.

Ambler, S. (2004): *Object Primer, The Agile Model-Driven Development with UML 2.0.*, Cambridge University Press.

AC3D. http://www.ac3d.org. Accessed August 2007.

ActiveWorld. http://www.ActiveWorld.com. Accessed August 2007.

Berner, S., Glinz, M., and Joos, S. (1999): A classification of stereotypes for object-oriented modeling languages. *Proc. UML'99 - The Unified Modeling Language. Beyond the Standard.* Second International Conference, Fort Collins, CO, USA, **1723**:249–264, Springer.

Blender. http://www.blender3d.org/cms/Home.2.0.html. Accessed August 2006.

Bille, W. (2007): Conceptual Modeling of Complex Objects for Virtual Environments, A Formal Approach. Ph.D. thesis. Vrije Universiteit Brussel, Brussels, Belgium.

Bille, W., De Troyer, O., Kleinermann, F., Pellens, B., Romero, R. (2004): Using Ontologies to Build Virtual Worlds for the Web. *Proc. of the IADIS International Conference WWW/Internet 2004 (ICWI2004)*, Madrid, Spain, **I**:683 - 690, IADIS PRESS, ISBN 972-99353-0-0.

Burdea, G.C., Coiffet, P. (2003): *Virtual Reality Technology*, Wiley-IEEE Press ISBN: 0471360899.

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

Chen, P. (1976): The entity-relationship model: Towards a unified view of data. *ACM Transactions on Database Systems*, 1(1):471–522.

Coninx, K., De Troyer, O., Raymaekers, C., Kleinermann, F. (2006): VR-DeMo: a Tool-supported Approach Facilitating Flexible Development of Virtual Environments using Conceptual Modelling, *Proc. of Virtual Concept 2006* Cancun, Mexico, Springer-Verlag, ISBN 2-287-48363-2.

De Troyer, O., Bille, W., Romero, R., Stuer, P. (2003): On Generating Virtual Worlds from Domain Ontologies. *Proc. of the 9th International Conference on Multi-Media Modeling*, Taipei, Taiwan, ISBN 957-9078-57-2, 279 – 294.

De Troyer, O., Kleinermann, F., Mansouri, H., Pellens, B., Bille, W., Fomenko, V. (2007): Developing semantic VR-shops for e-Commerce. *Virtual Reality* **11**(2): 89-106.

Dragicevic P. and Fekete J.D. (2004): Support for input adaptability in the ICON toolkit. *Proc. of the 6th international conference on multimodal interfaces*, State College - USA, 25-30.

Flanagan, D. (2001): *JavaScript: The Definitive Guide*, O'Relly.

Figueroa P., Green M. and Hoover H.J. (2002): InTml: A Description Language for VR Applications. *Proc. of Seventh international conference on 3D Web technology*, Tempe, USA, 15-20.

Fishwick, P.A. (2000): 3D behavioral model design for simulation and software engineering. *Proc. of the fifth symposium on Virtual Reality Modeling Language* (Web3D-VRML), 7-16. ACM Press, California, USA.

Fowler, M. and Scott, K. (1999): *UML Distilled: a brief introduction to the standard object modeling language*. Addison-Wesley Professional, second edition.

Flux player. http://www.mediamachines.com/ Accessed August 2007.

Geiger, C., Paelke, V., Reimann, C. and Rosenbach, W. (2002): A framework for the structured design of vr/ar content. *Proc. of the ACM symposium on virtual reality software and technology*, 75-82. ACM Press, Seoul, Korea.

Gonzalez, J.M., Vanderdonckt, J. and Arteaga, J.M (2006): Method for developing 3D User Interfaces of Information Systems. *Proc of the 6th International Conceference on Computer-Aided Design of User Interfaces CADUI'2006*, Bucharest-Romania, 85-100, Springer-Verlag, Berlin.

Gutschmidt, T. (2003): *Game Programming with Python, LUA and Ruby*. Course Technology PTR.

Hartman, J. and Wernecke, J. (1998): *The VRML 2.0 Handbook*. Addison-Wesley Publishing.

Halpin, T. (2001): *Information Modeling and Relational Databases: From Conceptual Analysis to Logical Design*. Morgan Kaufmann, first edition.

KIM, G., Kang, K., Kim, H. and Lee, J. (1998): Software engineering of virtual worlds. *Proc. of the ACM Symposium on Virtual Reality Software and Technology*, 131-138. ACM Press, Tapei, Taiwan.

Kang, K.C., Kim, G.J., Lee, J.Y. and Kim, H.J. (1998): Prototype=function+behavior+form. *ACM SIGSOFT Software Engineering Notes*, 23(4):44-49.

Kleinermann, F., De Troyer, O., Creelle, C., Pellens, B. (2007): Adding Semantic Annotations, Navigation paths and Tour Guides to Existing Virtual Environments. *Proc. 13th International Conference on Virtual Systems and Multimedia (VSMM'07)*, Brisbane, Australia.

Kleinermann, F., De Troyer, O., Mansouri, H., Romero, R., Pellens, B., Bille, W. (2005): Designing Semantic Virtual Reality Applications. *Proc. of the 2nd INTUITION International Workshop*, Senlis, France, 5-10.

Limbourg Q., Vanderdonckt J., Michotte B., Bouillon L., Florins M. and Trevisan D. (2004): UsiXML: A User Interface Description Language for Context-Sensitive User Interfaces. *Proc. of the ACM AVI'2004 Workshop "Developing User Interfaces with XML: Advances on User Interface Description Languages"*, Gallipoli - Italy, 20-25.

Maya Personal Learning Edition. http://www.alias.com/. Accessed on January 2006.

Martinez, J., Delgado Mata, C. (2006): A Basic Semantic Common Level for Virtual Environments. *International Journal of Virtual Reality*, **5**(3): 25 – 32.

MilkShape3D. http://www.swissquake.ch/. Accessed August 2007.

Murdock, K. L. (2002): *3ds max 5 Bible*. Wiley Publishing.

Navarre D., Philippe Palanque P., Bastide R., Schyn A., Winckler M., Nedel L. and Freitas C. (2005): A Formal Description of Multimodal Interaction Techniques for Immersive Virtual Reality Applications. *Proc. of the Tenth IFIP TC13 International Conference on Human-Computer Interaction*, Rome, Italy.

Object Management Group (OMG). MDA http://www.omg.org/mda/. Accessed August 2007.

Octaga player. http://www.octaga.com/. Accessed August 2007.

Palanque P. and Bastide R. (1994): Petri net based design of user-driven interfaces using the interactive cooperative objects formalism. *Proc. of Interactive Systems: Design, Specification, and Verification*, Carrara, Italy.

Palmer, I. (2001): *Essential Java 3D: Developing 3D Graphics Applications in Java*, Springer.

Pellens, B. (2007a): A Conceptual Modelling Approach for Behaviour in Virtual Environments using a Graphical Notation and Generative Design Patterns. Ph.D. thesis. Vrije Universiteit Brussel, Brussels, Belgium.

Pellens, B., Kleinermann, F., De Troyer, O. (2006): Intuitively Specifying Object Dynamics in Virtual Environments using VR-WISE. *Proc. of the ACM Symposium on Virtual Reality Software and Technology*, Limassol, Cyprus, 334-337. ACM Press, ISBN 1-59593-321-2.

Pellens, B., De Troyer, O., Kleinermann, F., Bille, W. (2007b): Conceptual Modeling of Behavior in a Virtual

Environment, *Special issue: International Journal of Product and Development,* **4**(6):626-645. Fischer X. and Bernard A. (eds). Inderscience Enterprises, ISBN 1477-9056.

Performer. http://www.sgi.com/. Accessed August 2007.

PlanIt 3D. http://www.planit3d.com/. Accessed August 2007.

Quake. http://en.wikipedia.org/wiki/Quake. Accessed August 2007.

Second Life. http://www.secondlife.com/. Accessed August 2007.

Seo, J. and Kim, G.J. (2002): Design Presence: A Structured approach to virtual reality system design, *Presence: Teleoperators and Virtual Environments*, **11**(4):378-403.

Southey, F. (1998): Ossa: A modelling system for virtual realities based on conceptual graphs and production systems. Master thesis. University of Guelp.

Southey F. and Linders J.G. (2001): Ossa: A conceptual modelling system for virtual realities. *Proc of the 9th international conference on conceptual structures,* Volume 2120 of Lecture Notes in Computer Science, 333-345. Springer-Verlag, California, USA.

Tanriverdi V. and Jacob R.J.K. (2001): VRID: A Design Model and Methodology for Developing Virtual Reality Interfaces. *Proc of ACM Symposium on Virtual Reality Software and Technology*, Alberta, Canada.

Unreal. http://www.unreal.com/. Accessed on August 2007.

Vanacken, D., De Boekc, J., Raymaekers, C. and Coninx, K. (2006): NiMMiT: a Notation for Modelling Multimodal Interaction techniques. *Proc of International Conference on Computer Graphics Theory and Applications*, Setubal – Portugal, 224-231.

Vince, J. (2004): *Introduction to Virtual Reality*. Springer, ISBN 1852337397.

Virtools Dev. http://www.virtools.com/. Accessed on August 2006.

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

Willans, J. (2001): Integrating behavioural design into the virtual environment development process. PhD thesis. University of York, York, UK.

Wilson, J.R. and Eastgate, R.M., and D'Cruz, M. (2002): *Handbook of Virtual Environments,* chapter 17, 353-378.

Walsh, A.E. and Sevenier, M. (2005): *Core Web3D*. Prentice Hall, Upper Saddle River, USA.

X3D toolkit. http://artis.imag.fr/Software/X3D/. Accessed August 2007.

Xj3D. http://www.xj3d.org/. Accessed August 2007.