On Generating Virtual Worlds from Domain Ontologies[*]

Olga De Troyer, Wesley Bille, Raül Romero, Peter Stuer
Research Group WISE, Department of Computer Science,
Vrije Universiteit Brussel, Pleinlaan 2, B-1050 Brussel, Belgium
OLGA.DETROYER@VUB.AC.BE, WBILLE@VUB.AC.BE, RAUL.ROMERO@VUB.AC.BE, PSTUER@VUB.AC.BE

Abstract
Virtual Worlds are computer-hosted visual environments that 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. Nowadays you need to be skilled in Virtual Reality (VR) technology to be able to develop such an environment. However, VR specialists are usually not experts in the domain for which the Virtual World need to be developed and vice versa. In this paper, we propose an approach that allows to generate a Virtual World from a domain ontology. A domain ontology describes the objects, their characteristics and their relationships that exist in the domain under consideration. In this way, a domain expert can specify the Virtual World at a high level and the most of the code for the Virtual Word can be generated. We believe that such an approach can open up the use of VR to a much broader community than nowadays.

1 Introduction

Virtual Reality (VR) is used in lots of different applications and for solving different problems. The filming industry, medical research and car industry all use some form of VR in solving different problems. Another well-known type of VR is 3D Computer games. 3D computer games have more or less become the standard in the gaming world. Also on the World Wide Web VR is appearing.

Currently, there are quite a number of different ways to develop VR applications. The available software can be divided into two major categories: toolkits and authoring systems. Toolkits (e.g. Total Havok, Vortex) are programming libraries that provide a set of functions with which a skilled programmer can create VR applications. Authoring systems (e.g. trueSpace, VRCreator) are complete programs with graphical interfaces for creating worlds without the need to resort to do detailed programming. However, in both cases skilled people are needed to develop a VR application. The authoring systems requires less programming skills but you still need to be skilled in VR technology to be able to build advanced applications. For both approaches, the problem is that the Virtual World you want to create must be expressed in terms of (low level) building blocks of the VR technology. This means that objects in the problem domain (e.g. a castle) need to be translated into a combination of VR

[*] This work was partially supported by the Institute for the Promotion of Innovation by Science and Technology in Flanders (IWT) in the context of the project OntoBasis.

primitives (such as cylinders, spheres, textures, …). None of the available VR development tools allow the developer to generate the Virtual World in terms of concepts and objects of the problem domain.

In this paper we describe an approach to generate Virtual Worlds based on high-level descriptions of the objects in the World to generate. These high level descriptions are specified by means of an ontology. In general, an ontology allows to define concepts (also called terms) as well as relationships between these concepts. The idea underlying our approach is that the (domain) objects and their properties needed in the Virtual World are defined and described in a (domain) ontology. From this domain ontology the Virtual World can be generate in a number of steps. Such an approach has several advantages. Firstly, the developer needs no longer to be an expert in VR technology because the World is specified at a conceptual level (by the ontology) and no longer at the implementation level. Secondly, the development time can be shortened (most of the code can be generated). And thirdly, maintenance will become easier because modification can be made at the conceptual level and propagated by the tool to the implementation. We will illustrate the feasibility of our approach by describing a first prototype. In this prototype DAML+OIL [Connolly et al., 2001] is used for representing ontologies, and the Vortex 3D engine from Critical Mass Labs [Critical Mass Labs, 2002] is used as the target VR software to generate code for.

The rest of the paper is structured as follows. In section 2 we explain our approach into more detail and motivate the choices made to implement the prototype. In the following section (section 3) we will introduce and explain the different ontologies used in the system and in section 4 we describe into more detail the actual generation process of the current prototype. In section 6 we discuss related work and in section 7 we present conclusion and discuss further work.

2 Overview of the Approach

Ontologies are very popular these days. They are considered as the solution for many problems that are related with terminology. E.g. someone can use the term ‘zip code’ while somebody else uses ‘postal code’. As a human being we know that both terms refer to the same concept but this is not obvious for a computer system. This kind of confusion can now be resolved by means of ontologies. If there exist an ontology that contains both terms and a relation that indicate that both terms are synonyms, then this kind of problem can be solved in an automatic manner. In its most simple form, we can say that an ontology is an abstraction of a computer-based lexicon, thesaurus, glossary or some other type of structured vocabulary, suitably extended with knowledge about a given domain. For more information about ontologies we refer to [Guarino & Giaretta, 1995] and [Gruber, 1993].

In this research we will not use ontologies to solve terminology problems, but to make software development, and more in particular the development of Virtual Worlds, easier. The goal of our approach is to generate code for a Virtual World that is described by means of a domain ontology (see figure 1). Ideally, the domain ontology should contain all information needed to generate the code for the Virtual World. Currently this is not yet the case, but we hope to reach this ideal situation after a number of research cycles (see also section 5 on future work).

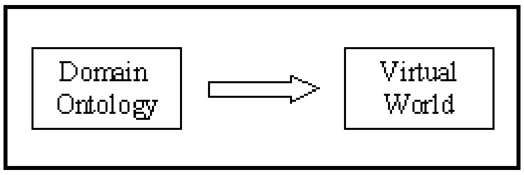


Figure 1: Generating a Virtual World from a Domain Ontology

The gap between the domain ontology and the current VR development software is too big to be captured in one step. Therefore it is broken down into a number of steps. First the Domain Ontology is converted into what we call a *Representable Domain Ontology*. This Representable Domain Ontology will describe how the objects in the Domain Ontology can be represented in the target VR development environment. In fact it contains the mapping from the domain objects into the primitive VR objects of the development software (Vortex objects in case of our prototype). In the next step, the Representable Domain Ontology is translated into a working Virtual Environment (C++ with Vortex objects in case of our prototype). See figure 2 for an illustration of these two steps.

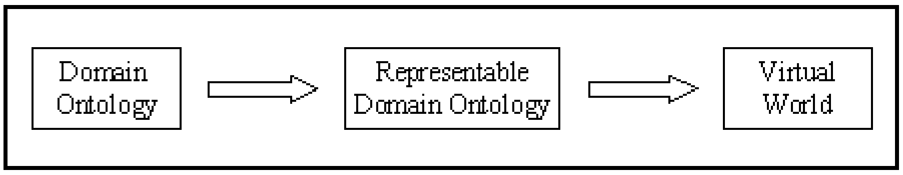


Figure 2: First decomposition of the generation process

The two steps are further described in section 4. But first, we define and describe the different ontologies that we need during the generation process. This is done in section 3. In the rest of this section we shortly describe the technologies chosen to implement the prototype.

In the prototype that we developed, we used the DAML+OIL language [Connolly et al., 2001] for representing the ontologies. DAML+OIL is a semantic markup language for Web resources. It is a combination of two languages DAML and OIL. The DAML language is being developed as an extension to XML ([XML, 1997], [Marchal, 1999]) and RDF ([Lassila & Swick, 1999], [RDF, 2002]). The latest release of the language provides a rich set of constructs with which to create ontologies and to mark-up information so that it is machine readable and understandable. The language has a clean and well-defined semantics. Several tools and data are available.
OIL [OIL, 2002] is a language built on a long history of research in description logics. Description Logic is a sub-field of knowledge representation and as such aims to provide a vehicle for expressing structured information and for reasoning with the information in a principled manner. OIL, which stands for Ontology Inference Layer, is an effort to produce a well defined language for integrating ontologies with web standards, in particular XML, XML Schema, RDF and RDF Schema. It is a web based representation and inference layer for ontologies using the constructs found in many frame languages and reasoning and formal semantics in description logics.
In December 2000, DAML and OIL were brought together as the DAML+OIL language specification. A DAML+OIL knowledge base is a collection of RDF triples, that is, a resource, a property and a literal. The relation between those three things is

as follows: the *"literal"* has *"property"* *"resource"*. As an example of this take the following: lets say that the resource is 'http://wise.vub.ac.be/WofSy/Head", the property is 'is physical representation' and the literal is 'Cylinder', then we can say: "Cylinder is the physical representation of the resource http://wise.vub.ac.be/WofSy/Head.
As VR development environment we opted for the Vortex 3D engine[1] from Critical Mass Labs [Critical Mass Labs, 2002]. This engine provides advanced physics, rigid-body dynamics and collision detection. It delivers fast stable physics simulation suitable for use in interactive 3D applications. It is possible to create sophisticated environments using Vortex.

3 The Ontologies

In the first step of the generation process, a Representable Domain Ontology is built for the given Domain Ontology. This Representable Domain Ontology contains the mapping of the domain objects into VR objects. In the case of our prototype this means a mapping from DAML+OIL objects into Vortex objects. To be able to describe this mapping we first need to describe which type of Vortex objects we can have and which type of DAML+OIL objects we can have. To do this, we will use two ontologies (also given in DAML+OIL syntax): the *DAML Ontology*, describing what kind of object types you can have in DAML+OIL and their relationships; and the *Vortex Ontology*, describing what kind of object types you can have in Vortex and their relationships. These two ontologies are described into more detail in section 3.1 and 3.2 respectively. Also the mapping mechanism used to map DAML+OIL objects into Vortex objects will be described using an ontology: the *Mapping Ontology.* This ontology together with the DAML Ontology and the Vortex Ontology form a kind of meta-level (see figure 3 – the arrows pointing from the Mapping Ontology to the DAML Ontology and the Vortex Ontology indicate that the Mapping Ontology uses concept defined in the two other ontologies). These ontologies are independent of the domain that we consider for generating the Virtual World. They are created only once. The DAML Ontology can be consider as the meta schema of DAML+OIL; the Vortex Ontology as the meta schema of the Vortex Engine; and the Represented-By Ontology as the meta schema of the mapping between DAML+OIL and Vortex. Someone may wonder why we use ontologies for this and not e.g. classical Meta schemas. The answer is simple. By using ontologies we have a single representation formalism for all the knowledge representation in the system that in addition offers the needed flexibility.

[1] Although the engine is not free to use, Critical Mass Labs provided us a free license for use in this project.

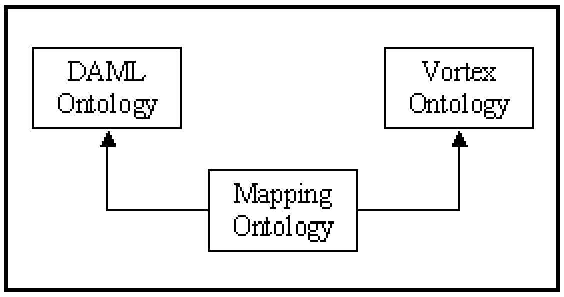


Figure 3: Ontologies representing the Meta level

To simplify the specification phase we will use a default representation (mapping) that can be used for all kind of objects. This default mapping will be captured in the *DefaultMapping Ontology*. This ontology can be seen as an instantiation of the Mapping Ontology, only containing a default mapping. The complete mapping for the target domain is given by the Representable Domain Ontology (introduced earlier and described in section 3.6). An overview of the different ontologies and their relationship to each other is given in figure 4 (also in this figure, an arrow indicates that one ontology is using concepts from the other ontology).

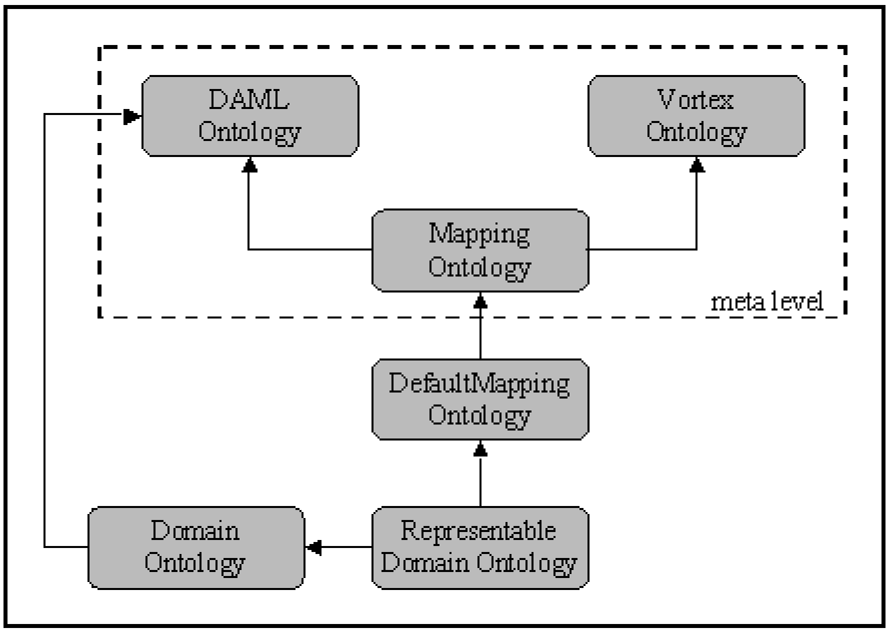


Figure 4: Overview of the different ontologies

Now, the different ontologies are explained into more detail in the next subsections.

3.1 The DAML Ontology

The ontology describing DAML+OIL is an existing ontology developed to be the normative reference on the precise syntax of the language constructs. In fact, it is the machine-readable RDF Schema definition of DAML+OIL. In this ontology we can for example see that a DAML class and a DAML data type are subclasses of the Class concept in the RDF Schema. The following piece of code is a sample taken from the DAML+OIL ontology, describing a DAML class.

```
<rdfs:Class rdf:ID="Class">
```

```
        <rdfs:label>Class</rdfs:label>
        <rdfs:comment>
            The class of all "object" classes
        </rdfs:comment>
        <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    </rdfs:Class>
```

The complete specification of this ontology can be found on the website of DAML [DAML+OIL Ontology, 2001].

3.2 The Vortex Ontology

As we mentioned before, the Vortex Ontology will give the entire description of the object types available in the Vortex engine. It contains information like the primitive object types that can be displayed using the 3D-engine and the different constraints and connections that are possible. To be able to generate code in a later stage we also need to know which parameters are needed in order to create instances of these object types. For example if we need to generate a sphere we will need to provide the radius, length and depth. We also need the type of the parameters in order to verify the input given by the developer later on. As an example we show how a sphere (a Vortex primitive) is described in the Vortex Ontology:

```
<daml:Class rdf:ID=”Sphere”>
    <rdfs:subClassOf rdf:resource=”#Object”/>
</daml:Class>
```

So a sphere is described as being an Object. The definition of the class Object is also given in the Vortex Ontology. All Objects are defined as being representable objects in the Virtual World.

We also have definitions of attributes in the Vortex Ontology like in the following example:

```
<daml:Class rdf:ID=”Xpos”>
    <rdfs:subClassOf rdf:resource=”#Attributes”/>
    <HasType
        rdf:resource=”http://www.w3.org/2001/XMLSchema#nonNegativeInteger”/>
</daml:Class>
```

Here, Xpos is defined as an Attribute of type nonNegativeInteger.

Using these kinds of definitions we can define the attributes needed by the different type of objects. As an example we give the description of an instance of a sphere.

```
<Sphere rdf:ID="SphereInstance">
    <HasAttributes rdf:resource="#Xpos" />
    <HasAttributes rdf:resource="#Ypos" />
    <HasAttributes rdf:resource="#Zpos" />
    <HasAttributes rdf:resource="#Color" />
    <HasAttributes rdf:resource="#Radius" />
    <HasAttributes rdf:resource="#Mass" />
</Sphere>
```

We see that an instance of a sphere has a position in a three dimensional space, represented by the x-position, the y-position and the z-position. A sphere also has a color, a radius and a mass.

Further on the Vortex Ontology contains descriptions of the connection types that can be use to connect primitive objects to each other to form composite objects. Vortex provides lots of different connection types like for instance ‘a ball and socket joint’.

The Vortex Ontology also contains concepts that are not representable, like for instance ‘gravity’ that has no direct physical representation in the Virtual World.
Therefore, the main concept in the Vortex Ontology is the class WorldConcept. All things that can exist in a Virtual World, whether or not they are representable, will be a subclass of the class WorldConcept.

3.3 The Mapping Ontology

As we mentioned before, this ontology describes the mapping between the two formalisms used, namely between the DAML+OIL concepts and the Vortex concepts. This ontology contains only one class describing the properties of this mapping. The class is declared as follows:

```
<daml:Class rdf:ID=”Representation”>
    <rdfs:subClassOf>
        <daml:Restriction daml:mincardinality=”1” daml:maxcardinality=”1”>
            <daml:onProperty rdf:resource=”#Source”/>
        </daml:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <daml:Restriction daml:mincardinality=”1” daml:maxcardinality=”1”>
            <daml:onProperty rdf:resource=”#Target”/>
        </daml:Restriction>
    </rdfs:subClassOf>
</daml:Class>
```

As we can see, a Representation has exactly one Source and exactly one Target property. Source and Target are defined as follows:

```
<daml:ObjectProperty rdf:ID=”Source”>
    <rdfs:range rdf:resource=”http://www.daml.org/2001/03/daml+oil#Thing”/>
</daml:ObjectProperty>
<daml:ObjectProperty rdf:ID=”Target”>
    <rdfs:range
rdf:resource=”http://wise.vub.ac.be/WofSy/VortexOntology.daml#WorldConcept”/>
</daml:ObjectProperty>
```

Source is an ObjectProperty that has a Daml:thing as resource. Note that everything in DAML is a Daml:thing. In this way, every DAML concept can be used as a value for the Source property. On the other hand, the Target, which is also an ObjectProperty, has as resource a WorldConcept from the Vortex Ontology. This allows us to use any concept from the Vortex ontology as a value for the Target property. In this way, the

Representation class can be used to map any object from DAML onto any object from Vortex.

3.4 The DefaultMapping Ontology

As explained earlier, it is handy to have a default representation. This default can be used for fast prototyping the desired Virtual Word; the developer only needs to specify presentations for the most important objects, for the others the default representation will be used. Here, we have chosen to take a Sphere as default representation, but this is only a matter of personal taste: a cone or a box is equally good. During the generation of the Virtual World, the default representation will be used for an object unless there is a representation specified by the developer in the Representable Domain ontology.

The following definition defines a sphere as the default representation. Note that the default representation is declared as an instance of the Representation class from the Mapping Ontology (‘rprs’ is the prefix referring to the namespace declared as being the Mapping Ontology):

```
<rprs:Representation rdf:ID="ClassRepresentation">
        <rprs:Source rdf:resource="http://www.daml.org/2001/03/daml+oil#Class" />
        <rprs:Target
                rdf:resource="http://wendy.vub.ac.be/~wbille/VortexOntology.daml#Sphere"
/>
</rprs:Representation>
```

3.5 The Domain Ontology

The Domain ontology is the ontology describing the Virtual World someone wants to generate. It is supposed to contain all domain objects that are required in the Virtual World. For the moment, the developer needs to create this ontology. However, in the future we like to consider the possibility to use existing ontologies that describe the problem domain for which the Virtual World need to be created (more on this in section 5 about further research).

To describe composite objects, the collection concept of DAML can be used. Suppose we want to describe a personal computer as being a composition of a screen, a keyboard, a mouse and a computer unit, we can do this the following way:

```
  <daml:Class rdf:ID="PersonalComputer">
      <daml:unionOf rdf:parseType="daml:collection">
          <daml:Thing rdf:about="#Screen"/>
          <daml:Thing rdf:about="#Keyboard"/>
          <daml:Thing rdf:about="#Mouse"/>
          <daml:Thing rdf:about="#ComputerUnit"/>
      </daml:unionOf>
  </daml:Class>
```

Note that there also has to be a description of the objects in a collection. So a definition of Screen, Keyboard, Mouse and ComputerUnit is also needed (which may be composition as well).

3.6 The Representable Domain Ontology

As already explained the Representable Domain ontology describes the mapping from domain objects into Vortex objects. Therefore it contains instances of the Representation class described in the Mapping Ontology linking objects from the Domain Ontology to objects from the Vortex Ontology. Note that it is not necessary to have a representation for each object in the Domain Ontology as we have the default representation that can be used defined in the DefaultMapping Ontology (see section 3.4).

How this Representable Domain Ontology is created is explained in the next section. In this section also a fragment of such a Representable Domain Ontology is given.

4 The Generation Process

As explained in section 2, the generation process is decomposed into two steps. In the first step the Representable Domain Ontology is created for the given Domain Ontology, and in the second step code is generated from this ontology for the target VR environment. The first step is realized by the *Domain Mapper*, the second step by the *World Creator*. In the next subsections, these two tools are explained into more detail.

4.1 The Domain Mapper

The Domain Mapper is software that can be used by the developer to produce a Representable Domain Ontology for a given Domain Ontology. Input is the Domain Ontology and possible existing Representable Domain Ontologies that are suitable for the given domain. In this way, existing representations (available in those Representable Domain Ontologies) can be reused and adapted if desired. If no suitable representation is available the developer can specify one in an interactive way.

Below is a fragment of the possible output of the Domain Transformer for the following small and simple Domain Ontology:

```
…
<daml:Class rdf:ID=”Wall”/>
<daml:Class rdf:ID=”Tower”/>
<daml:Class rdf:ID=”Planet”/>
<daml:Class rdf:ID=”Pyramid”/>
…
```

Fragment of the generated Representable Domain Ontology:

```
...
<RDFNsId1:Representation rdf:ID='TowerRepresentation'>
	<RDFNsId1:Source rdf:resource='http://sch472.vub.ac.be/domain.daml#Tower'/>
	<RDFNsId1:Target
rdf:resource='http://wise.vub.ac.be/WofSy/VortexOntology.daml#Cylinder'/></RDFNsId1:Re
presentation><RDFNsId1:Representation rdf:ID='WallRepresentation'>
	<RDFNsId1:Source rdf:resource='http://sch472.vub.ac.be/domain.daml#Wall'/>
```

```
    <RDFNsId1:Target
rdf:resource='http://wise.vub.ac.be/WofSy/VortexOntology.daml#Box'/></RDFNsId1:Repres
entation>
<RDFNsId1:Representation rdf:ID='PyramidRepresentation'>
    <RDFNsId1:Source rdf:resource='http://sch472.vub.ac.be/domain.daml#Pyramid'/>
    <RDFNsId1:Target
rdf:resource='http://wise.vub.ac.be/WofSy/VortexOntology.daml#Cone'/></RDFNsId1:Repre
sentation>
<RDFNsId1:Representation rdf:ID='PlanetRepresentation'>
    <RDFNsId1:Source rdf:resource='http://sch472.vub.ac.be/domain.daml#Planet'/>
    <RDFNsId1:Target
rdf:resource='http://wise.vub.ac.be/WofSy/VortexOntology.daml#Sphere'/></RDFNsId1:Rep
resentation>...
```

As you can see, for each object described in the Domain Ontology, we have an instance of the Representation class. You can also see that each object is mapped onto some (in this case, simple) physical representation in Vortex.

If a Representable Domain Ontology is generated, the next step is to generate the Virtual World itself. The World Creator does this.

4.2 The World Creator

The World Creator software transforms the Representable Domain Ontology into a working Virtual Environment. For the moment, we need to ask the developer to give additional information about the objects in the domain during this generation process, e.g. the value of attributes. We will eliminate this in the future as much as possible (see section 5 on future research). However, the difference with the current authoring tools is that the developer does not need to specify the attribute values in terms of spheres, cylinders, boxes and so on, but in terms of domain objects; he will be asked for the position of the tower, the color of the wall, and so on. In this way, the developer really works in his knowledge domain and not in the domain of VR technology.

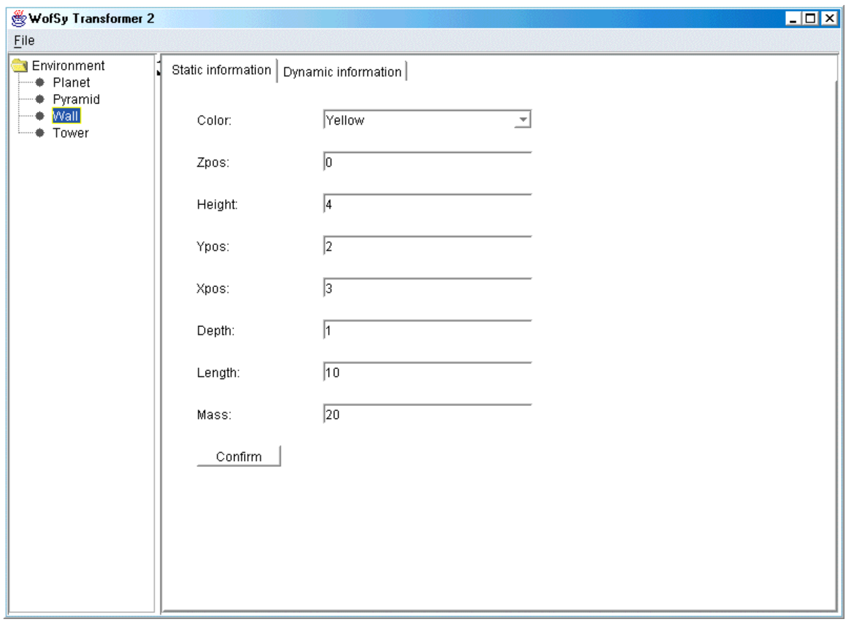
The World Creator not only creates the Virtual World, but also an ontology containing a complete description of the generated World. In this way, the constructed Virtual World can be reconstructed without the need to re-generate it. We only have to take this ontology and the World Creator can re-generate the complete world without having to ask the developer for the parameters needed for the objects.

During the implementation of the prototype we started with the generation of the simplest case, those of a statical Virtual World with only non-composite objects. Next, we experimented with adding dynamic behavior to the World. To have dynamic behavior in a world (like moving objects in the world) the C++ code for the world has to contain some procedures implementing that behavior. To be able to generate such procedures, we needed extra information. Again, for the moment, the developer can specify this in an interactive way during the generation process. At the moment of writing, it is e.g. possible to attach an initial velocity and an initial angular velocity to an object, and the developer can declare whether the behavior has to be simulated continuous or if it has to be activated by pressing a key (even-driven). When there is dynamics in a world, there can be some collisions. Therefore, the generated code also takes care of collisions. Here we use the Vortex Collision detection.

Note that this is only a little part of possible dynamics in a world. Other things we can do in future are e.g. to put forces on objects, and putting forces to a specific point of an object. However, instead of elaborate on the dynamics, we preferred to first concentrate on the issue of generating Composite Objects.

Composite objects are objects build up from other simple or composite objects. Objects can be connected to each other by using some of the constructs that Vortex provides us. When an object is described as composite in the Domain Ontology by using the DAML collection construct, then the World Creator will take care that all the representations of the components are connected to form the whole. The developer will be asked for the type of connection he wants to use and for the properties of the connection. Some examples of connections are 'the ball and socket joint', 'the hinge joint', and 'the spring joint'. For each of these joints, properties can be set.

Figure 5 shows a screenshot from the World Creator. Figure 6 give an example of a generated Virtual World.



Color:	Yellow
Zpos:	0
Height:	4
Ypos:	2
Xpos:	3
Depth:	1
Length:	10
Mass:	20

Figure 5: Screenshot taken from the World Creator

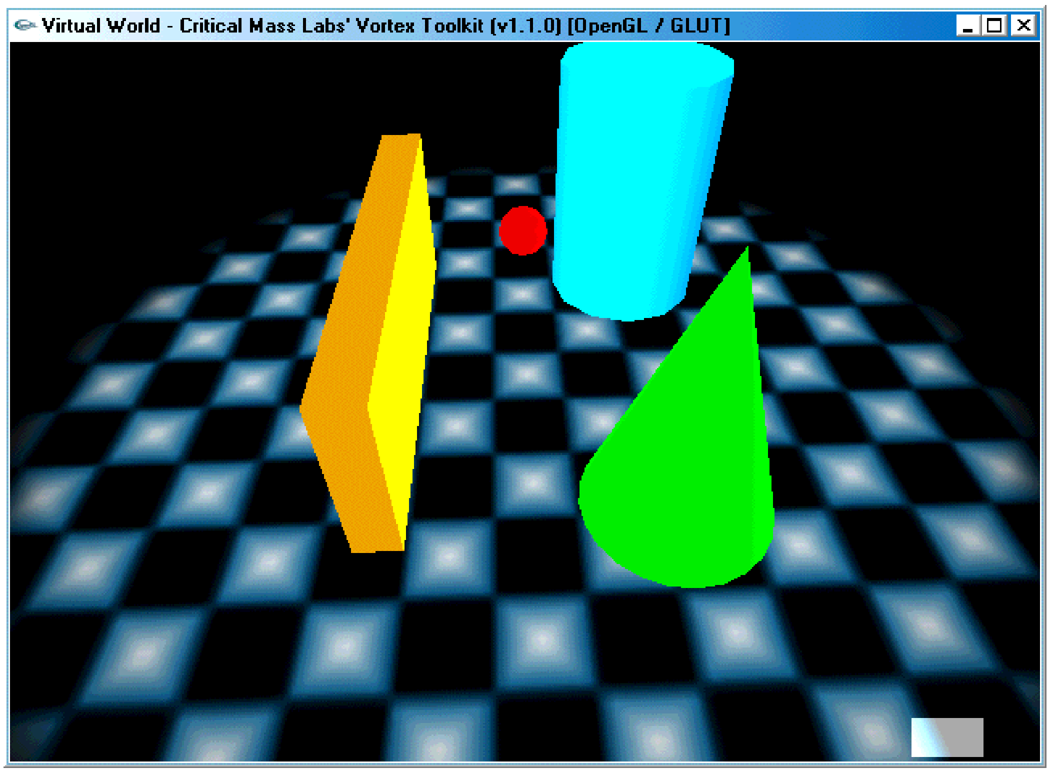

Virtual World - Critical Mass Labs' Vortex Toolkit [v1.1.0] [OpenGL / GLUT]

Figure 6: Example of a generated world

5 Related Work

The lack of high-level design methodologies for VR development has also been addressed in [Tanriverdi & Jacob, 2001] with the presentation of VRID (Virtual Reality Interface Design). In this paper, four key components are identified when designing VR interfaces: 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.

James Willans and Michael Harrison [Willans & Harrison, 2001] in the HCI group at the York University (UK) 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. The Marigold toolset is an attempt to find easier ways to design virtual environments like flight simulators where it is important to make the interaction between the user and the environment as realistic as possible. Abstract models are constructed using the Flownet modeling formalism. The toolset provides a means of automatically verifying the abstract model for desirable properties and supports a transition between the abstract model(s) and an implementation prototype of the model(s). The approach this group follows to support the modeling of requirements for virtual environments is presented in [Smith & Duke, 2000] and [Willans, Smith & Harrison, 2001]. Four important requirements are examined when designing virtual environments: appearance, decomposition and behavior of the objects and the user behavior through interaction techniques. Scenarios are used to identify the requirements and the formalism stated is based on a requirements tree that represents the different requirements identified.

6 Conclusions and Future Work

In this paper, we have proposed and described an approach to generate Virtual Worlds from high-level descriptions given in the form of an ontology. We believe that such an approach can open up the use of VR to a much broader community than nowadays. The objects needed in the Virtual World, their relationships and their properties can be described in term of the concepts used in the problem domain. Therefore, the main specifications can be made by a domain expert and currently only some minor help from a VR specialist is needed during the transformation process. We expect to be able to reduce this input of the VR specialist to a minimal by refining the information in the domain ontology and by using upper level ontologies, i.e. it must be possible to derive from domain knowledge and general common sense knowledge a suitable representation for an object in the Virtual World. E.g. if we know that a ball is round and is a 3D object we can derive that a sphere might be a suitable representation.

To prove the feasibility of our approach we implemented a prototype. The prototype is still very simple and can improved in different ways:

- More information can be derived from ontologies:
  In the current version, quite some information need to be specified interactively during the generation process. In principle it must be possible to extract or derive most of this from the Domain Ontology or from some upper level ontology. E.g. the size and colour of an object can easily be specified as properties of the object in the Domain Ontology; to a certain extent the same applies for the behaviour of the objects. Also the way composite objects are composed of other objects can be specified into more detail in the ontology such that less information needs to be asked during generation.
- More advanced Virtual Worlds can be generated:
  The worlds that can be generated for the moment are rather simple. Several extensions are possible such as more sophisticated behaviour, more complex objects, more sophisticated interaction possibilities, introduction of materials and the consequences of this (when a ball of rubber collides with a wall this is different than when the ball was made of glass), support for scaling, generation of objects using some deviation (e.g. generating 100 people according to some standard deviation such that we have thin, fat and normal persons, small and tall persons, …), and so on.

Currently we start from a domain ontology that is designed especially for our tool. Creating such an ontology for a domain may be a lot of work, therefore it may be interesting to investigate if it is possible to extract the necessarily information from one or more ontologies which are already in existence for the domain under consideration.

7 References

[Berners et al., 2001] T. Berners Lee, J. Hendler and O. Lassila, “The Semantic Web: A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities”, in ‘The Scientific American’, May 2001.

[Connolly et al., 2001] D. Connolly, F. van Harmelen, I. Horrock, D. McGuinness, P.F. Patel-Schneider, L. A. Stein, ”Annotated DAML+OIL Ontology Markup”, W3C Note 18 December 2001, http://www.w3.org/TR/daml+oil-walkthru/.

[Critical Mass Labs, 2002] http://www.cm-labs.com, CMLabs Simulations, Inc., 2002

[DAML+OIL Ontology, 2001] http://www.daml.org/2000/12/daml+oil.daml

[Guarino & Giaretta, 1995] N. Guarino and P. Giaretta, "Ontologies and knowledge bases: towards a terminological clarification", in Towards Very Large Knowledge Bases: Knowledge Building Knowledge Sharing (ION Press), 1995, pp. 25-32.

[Gruber, 1993] T. R. Gruber, “A translation approach to portable ontologies“, Knowledge Acquisition, 5(2), 1993, pp. 199-220.

[Lassila & Swick, 1999] O. Lassila and R. Swick, “Resource Description Framework: Model and Syntax Specification”, W3C Recommendation, World Wide Web Consortium, Cambridge, 1999.

[Marchal, 1999] B. Marchal, “XML by example”, Que 1999 (ISBN 0-7897-2242-9).

[OIL, 2002] http://www.ontoknowledge.org/oil/, On-To-Knowledge, 2002.

[RDF, 2002] http://www.w3.org/RDF/, W3C, 2002.

[Smith & Duke, 2000] Shamus P. Smith, David J. Duke, “Binding Virtual Environments to Toolkit Capabilities”, EUROGRAPHICS Volume 19 (2000), eds. M. Gross and F.R.A. Hopgood

[Tanriverdi & Jacob, 2001] Vildan Tanriverdi and Robert J. K. Jacob, “VRID: A Design Model and Methodology for Developing Virtual Reality Interfaces”, Proc. ACM VRST 2001 Symposium on Virtual Reality Software and Technology, ACM Press, Banff, Canada, 2001.

[Willans, Smith & Harrison, 2001] James S. Willans, Shamus P. Smith and Michael D. Harrison, “Using scenarios to identify the design requirements of virtual environments”, Technical Report YCS 333, University of York, 2001.

[Willans & Harrison, 2001] James S. Willans and Michael D. Harrison, “A toolset supported approach for designing and testing virtual environment interaction techniques”, International Journal of Human-Computer Studies, 55(2), pp. 145-165, 2001.

[XML, 1997] http://www.w3.org/XML/, W3C, 1997-2002