Detecting Inconsistencies in the Design of Virtual Environments over the Web using Domain Specific Rules

Haïthem Mansouri
Vrije Universiteit Brussel
Pleinlaan 2
1050 Brussels
+32 2 629 5713

Frederic Kleinermann§
Vrije Universiteit Brussel
Pleinlaan 2
1050 Brussels
+32 2 629 3504

Olga De Troyer²
Vrije Universiteit Brussel
Pleinlaan 2
1050 Brussels
+32 2 629 5713

ABSTRACT

Nowadays, 3D Virtual Environments (VEs) are being used over the web for various purposes such as education, collaborative working or social networking. Unfortunately, the development process of such environments remains a demanding task, often accessible only to VE experts despite the availability of a number of Virtual Reality (VR) authoring tools. On the other hand, VE experts are seldom domain experts. This implies that their knowledge on specific domains can most of the time be limited. This could lead to design errors or, in most cases, longer development times and efforts as the development process become an iterative one involving many revisions. One way of accelerating this process is by making it possible to capture a specific knowledge of a domain and later use this knowledge to automatically check that the design of the VE meets the requirements of the domain. This way, we ensure the conformity of the VE to the requirements of the domain for which it is being developed and by extension also to the customer’s requirements. As a result, development times and efforts can significantly be shortened, while reducing the likelihood of error making. This paper describes an extension to an existing approach called VR-WISE that focuses on reducing development times and efforts of VEs using domain oriented terminology and ontologies.

Categories and Subject Descriptors
H.5.1 [Information Interfaces and presentation] Multimedia Information Systems - artificial, augmented, and virtual realities, Evaluation/methodology

General Terms
Design

Keywords
Virtual Environments, Virtual Reality, Domain Specific Rules, Semantics, X3D

1 INTRODUCTION

Nowadays, 3D Virtual Environments (VEs) for the web are being used for various purposes including education, collaborative working or social networking. Examples of such VEs are applications like Vivaty [Vivaty 2009] and SecondLife [SecondLife 2009]. However, the design of such VEs remains a challenge as they need to address different domains and they must be usable by a large audience. The challenge is bigger for non-VE experts like web designers who want to develop these applications and meet the customer’s requirements. Beside this, they also need to generate quickly new VEs for different domains. Authoring tools such as 3ds Max [3ds Max 2009] and Blender [Blender 2009] can help designers to rapidly create virtual models. But in most cases, the design of rich VEs will involve the use of more than one authoring tool. As a result, the design of VEs is often left to VE experts (or at least designers with enough knowledge about how to build a VE) to develop the VE. This leads to an iterative time consuming design process where a VE is modified several times until it meets the customer’s requirements. To overcome this problem, a number of approaches (see related section) have focused on making the design process of VEs more intuitive by involving the domain expert(s) (who may be the customer) at an early stage of the design. While these approaches can contribute to reducing development times and costs by bringing down the usual number of iterations needed, it does not, however, remove the inherent complexity related to the building of a VE. In all these approaches (see related section), VE experts are still involved at some point in the design. Furthermore, they are still expected to acquire a firm grasp in understanding the complexities of a specific domain before designing VEs for it. This is error-prone as no tool exists to correct them or highlight incoherent parts of the design according to the domain to which they are developing a VE. An example of this can be seen in Urban Design where, for instance, houses and buildings can have certain height limitations in some residential areas. Unless the properties of a specific domain are thoroughly researched, a VE expert might not be aware of such constraints, but a domain expert or a designer coming from that domain will have the necessary knowledge (or at least must pay attention to that).

In this paper, we are introducing a new approach that captures the semantic rules of a domain and used them to check the design of a VE for a certain domain before the actual code of the VE is generated. As a consequence, there are fewer chances to make domain specific mistakes leading to reduced development times and efforts.

The paper is structured as follows: Section 2 will introduce the VR-WISE approach as the work presented in this paper is based on that approach. Section 3 will explain the extension made to the VR-WISE approach in order to capture the semantics rules of a
domain. Section 4 will describe the tool which is implementing the approach presented in this paper and section 5 will provide some case studies. Section 6 will conclude the paper with a summary, limitations and future work.

2 BACKGROUND

The work presented here is done in the context of a more general approach, called VR-WISE, which is developed to support the development of Virtual Environments (VEs). This work has already been published in different papers and journals [De Troyer et al. 2003] [De Troyer et al. 2008] [Kleinermann et al. 2005]. We refer the reader to these journals.

The aim of the approach is to facilitate the development of VEs by including an explicit conceptual modelling phase into the overall development process. Conceptual modelling is the activity that creates technology independent models for the system to be constructed. The different stakeholders can use these models as a basis for discussing the design, but the models can also be used as the input for the implementation phase. VR-WISE enables the designer to specify a VE through such conceptual models. The VR-WISE approach shortens the development process of VEs by using an explicit Conceptual Specification phase in the development life cycle of a VE application and by using the terminology of the domain. The terminology of the domain in question is captured by either using an existing ontology of the domain or by asking domain experts to define the concepts of their domain. The VR-WISE approach has also developed and defined modelling concepts that are related to the building of a VE application and that are common to all VR applications. For instance, modelling concepts related to the positioning of objects like left-of, right-of; modelling concepts for behaviour like triggers, move and so on [Pellens 2007] [Pellens et al. 2008]. Using these modelling concepts and the terminology of the domain, the designer can then create a Conceptual Model of the VE application using the terminology from the application domain. During conceptual modelling, there is no need to consider implementation details. Hence, no real VR background knowledge is needed to create these conceptual models. VR-WISE is a model-driven approach. This means that the conceptual specifications (i.e. implementation independent models) are translated into implementation dependent models and ultimately into implementation code. Because this transformation/implementation should be (semi-) automatic, this implies that the expressive power of the different modelling concepts must be sufficient to allow (to a certain degree) for this automatic implementation phase. Currently, the VR-WISE approach has concentrated on the conceptual specification of the scene, the objects, behaviours and interaction.

Figure 1 shows that the VR-WISE approach consists of three main steps: the specification, mapping and generation steps. The specification step covers the design phase. Here, the VE is modelled at a high level using the domain concepts. Modelling at a high level means that the designer does not need to consider implementation details at this stage. Once the concepts have been created, it is possible to derive instances from them. Note that these instances are given attributes and added to the virtual world. In other words, the instances represent the objects populating the virtual world. This last step is referred to as world specification because these instances will be used to populate the virtual world. World specification describes how these instances are related, where they are placed in the world, how to interact with them and what behaviour they may have. Concepts can be made as generic as needed in order to allow them to be reused to create a variety of different virtual environments that share the same domain. It is typical to reuse concepts created by a domain expert as she or he will have the best knowledge about a particular domain.

The mapping stage is used to specify how the domain concepts and instances will be represented in the virtual world. This is done by mapping the concepts and instances defined in the previous step onto VE primitives (e.g. 3D spheres and boxes), constrains and so on. The generation stage will generate the world in a format such as X3D [X3D 2009] using the previous stages.

Although domain experts are involved in this approach and this may reduce the time to build a VE, it still requires a VE expert at some point to build the final VE (like for instance defining the mappings). As a result, it is still error-prone. This why we believe that extending the VR-WISE approach by capturing domain rules from the domain expert and by providing a way to formalise them so that they can later be used by a reasoning engine, will facilitate the task of the designer as his design can then be checked using the rules of the domain before the actual VE code is generated (in our case X3D code). For instance in the case of a virtual city, we can imagine that different rules from the domain of Urban Design can be captured in order to help VE expert to create virtual cities that meet the requirements of the domain of Urban Design. Such rules can contain, for example, the minimum and maximum values for street and pavement widths, maximum building heights in some residential areas, the positioning of street lamps, etc. Another example could be a 3D representation of a virtual factory where machines need to be placed with sufficient space in certain areas in order to guarantee a safe and free movement of operators.

Having the ability to capture these rules from a domain expert during the conceptual modelling step and use them to verify the “correctness” of a specific world during its creation would have many advantages such as ensuring the creation of correct worlds according to the domain and speeding up the design process. In the following section, we present an extension to VR-WISE approach that allows us to do precisely this.

3 Capturing Domain Specific Rules

The extension made to the VR-WISE approach is illustrated in Figure 2 where a module has been added in step one to indicate
that the domain rules need to be captured from the domain expert during the specification phase. These rules are then used to check the design of VE according to a domain (see figure 2).

Figure 2. Domain Specific Rules Module

For the extension to VR-WISE approach, we have developed two main stages namely Capturing the Virtual Environment knowledge and Formalization of Semantic Rules. We will now describe them.

### 3.1 Capturing the Virtual Environment knowledge

In order to reason about a particular VE, it is necessary to first capture its attributes and relationships between the different objects it is made out of. As the VR-WISE approach already uses ontologies expressed in the Web Ontology Language (OWL), we can already capture simple semantic relations between the concepts and their attributes. For instance, we can deduce semantic relations like is-part-of or is-a relation. Ontologies [Pittarello et al. 2006] are a natural choice for capturing and storing knowledge about a particular domain.

The type of information captured as well as the way it is associated with the virtual world are also important. This can determine, for instance, the accuracy of a specific reasoning engine or the quality of its results. For Cavazza [Cavazza 1998], it is necessary to simultaneously access both concrete and abstract information given for a particular virtual world. Concrete information reflects the direct visual representations of the scenes, while the abstract information is needed to master the complexity of the scenes and their dynamic behaviours.

Beside the information used for representing objects like height, width, depth and positions, we also want to capture other types of information that may be more abstract such as weight, price and so on. To capture this information, we have introduced the notion of semantic annotations.

#### 3.1.1 Semantic Annotations

In order to capture abstract information on a specific domain, we have extended the VR-WISE approach to incorporate the notion of annotations. Semantic Annotations are pieces of semantic information that a user can add to specific objects in a world during its creation. For example, the history of an object, its brand, the specific category it might fall under, etc. can all be entered in the form of semantic annotations.

The addition of semantic annotations is particularly important in order to fully capture domain specific information and the rules of a domain. Indeed, object properties alone are not sufficient for this purpose as they mainly deal with physical object representation information. For instance, an Urban Design expert may want to create rules that govern the maximal allowable heights of buildings in function of their location. In our approach, this last attribute, not being part of the property of an object, can be added in the form of a semantic annotation.

### 3.1.2 Semantic Rules

The VR-WISE approach provides already some high-level relations like spatial relation (left-of, right-of, etc.), relations between concepts (is-a and part-of), relations specific to complex objects with their constraints [Bille 2007]. These relations can be formalized and used by a reasoning engine to reason about the design. But these semantic rules are more dedicated to the scene and the physical relationships of the objects populating it. For this reason, we have also extended the approach so that more abstract semantic rules can be introduced by a domain expert. These rules are specific to a domain and can then provide a way to check the design of a VE more from a domain point of view. Note that these semantic rules need to be defined once for a domain and can then be used for each VE being designed for that domain.

Even though OWL offers a reasonable trade-off between expressiveness and decidability, its efficiency to reason on large-scale ontologies involving a big number of individuals can be questionable. Furthermore, OWL cannot be used efficiently to model certain application domains. While the latter limitation can be overcome by using Semantic Web-enabled rule languages such as the Rule Markup Language (RulML) or the Semantic Web Rule Language (SWRL), the former is more of a challenge. However, this can be overcome by using Frame-Logic (or F-Logic) [May 2006]. F-Logic has also the advantage to follow the Object-Oriented paradigm which is used in the VR-WISE approach. Note that instances in the VR-WISE approach are the actual objects populating the virtual world. F-Logic allows us to treat concrete and meta-level descriptions the same way for query purposes. F-logic also provides support for most aspects of OWL Full, which gives us flexibility for meta-modelling if needed. Note that this is not yet used by our tools. For all of these reasons, it was decided that F-Logic will be used to formalize the semantic rules in our approach. More details on frame-Logic can be found in [May 2006] [Ontoprise 2006].

In order to be able to reason about the design, these rules must be formalized. The next section introduces this formalization.

### 3.2 Formalization of Semantic Rules

To illustrate the F-logic formalization, we will now describe some of the semantic rules that have been formalized with F-Logic. More details on this formalization can be found in [Bille 2007].

We start by defining the entity concept.

**Definition 1:** A concept is a subclass of the highest level entity object. Concepts can have properties, annotations as well as hierarchy relations.
Further, we define an instance as being member of the class concept. As such, the instance class inherits properties and annotations of the parent class. Instances are actual ‘physical’ representations of objects in the virtual environment. Therefore, some information is required in order to position them correctly inside the world. This is done through one of the following direct attributes hasPosition and hasOrientation or indirect ones: hasSpatialRelation and hasOrientationBySide. Indirect positioning methods mean that an object is positioned in the world relative to another object.

**Definition 2:** An instance object is defined as a member of the class concept. In addition to inherited methods, instances have positioning information or relations.

instance:concept[hasPosition ⇒ position;  
    hasOrientation ⇒ orientation;  
    hasOrientationBySide ⇒ orientationBySide;  
    hasSpatialRelation ⇒ spatialRelation;  
    isPartOf ⇒ instance;  
    consistsOf ⇒ instance].

**Definition 3:** Annotations and Properties are defined as having a name (or keyword) and value.

annotation[annotationKeyword ⇒ annotationValue].  
property[propertyName ⇒ propertyValue].

Finally, we define the Static Structure Descriptors and relations as follows.

**Definition 4:** Static Structure Descriptors are defined as:

position:::StaticStructureDescriptor[X_Position ⇒ float,  
    Y_Position ⇒ float, Z_Position ⇒ float].  
orientation:::StaticStructureDescriptor[X_Angle ⇒ float,  
    Y_Angle ⇒ float, Z_Angle ⇒ float].  
orientationBySide:::StaticStructureDescriptor[OrientationSource ⇒ instance,  
    OrientationSourceOrientation ⇒ orientation,  
    OrientationTarget ⇒ instance,  
    OrientationTargetOrientation ⇒ direction].  
orientationByAngle:::StaticStructureDescriptor[  
    OrientationSource ⇒ instance,  
    OrientationSourceOrientation ⇒ orientation,  
    OrientationTarget ⇒ instance,  
    OrientationTargetOrientation ⇒ direction,  
    OrientationAxis ⇒ axis].  
spatialRelation:::StaticStructureDescriptor[  
    spatialRelationSource ⇒ instance,  
    spatialRelationTarget ⇒ instance,  
    spatialRelationDirection ⇒ direction,  
    spatialRelationReference ⇒ object].

direction[string] ⇒ [left: right; front: back;  
    above: below].  
axis[string] ⇒ [front back: above below: left right].

We have also defined formal definition of semantics for complex objects based on the work of Bille [Bille 2007]. To specify that a concept is a component of some other concept, a part of property is used.

Let a and b be two concepts, thus a::concept and b::concept. The fact that a is part of b is expressed by adding a property isPartOf to the concept definition of a: a[isPartOf ⇒ b].

In this way, we can state that if a concept a is part of a concept b, this implies that b consists of the concept a:

Let a and b be two concepts, thus a::concept and b::concept. The fact that a is part of b implies that b consists of a:  
b[consistsOf ⇒ a] ← a[isPartOf ⇒ b].

The inverse is also true: a[isPartOf ⇒ b] ← b[consistsOf ⇒ a].

**4 Tool**

In order to test our approach, we extended the tool “OntoWorld” [Kleinermann et al. 2005] [De Troyer et al. 2007] [Pellens 2007] [Wesley 2007] that implements VR-WISE with our approach. OntoWorld is a software tool that implements the VR-WISE approach. This tool is built in C# and on the .NET platform. OntoWorld has been designed to generate virtual environments from explicit semantic descriptions using ontologies.

We are using a three-layered approach for storing information. We differentiate between a meta-level description, a domain description and a world description. The meta-level description is an ontology used to describe the terminology used in the domain and world. This ontology will define concepts such as annotation, relation, position, orientation, etc. The meta-description is a generic ontology used to define the grammar of the world description scheme.

As its name suggests, the domain description holds information about a particular domain and is generally created by a domain expert. An example of a domain ontology can be the description of a city. Such ontology would hold information about the fact that a city is comprised of buildings - some of which would have a residential purpose, while other would be commercially oriented, for instance - streets, road signs, vehicles, parks, etc. The information stored at this level can be very generic in order to guarantee openness and re-usability.

Finally, the world description holds information about specific objects in the virtual environment. As the world description inherits classes and attributes from the domain description, it can become rich in information if multiple source ontologies are used.
Figure 3 shows an extract from the Meta description ontology defining an Annotation. We can see that an annotation is defined as consisting of a keyword and a corresponding value. The information gathered in the different ontologies is later fed into a reasoning engine to build the knowledge base.

![Image of OWL representation](rdf.png)

**Figure 3. OWL Representation**

For our description purposes we have restricted ourselves to the use of the OWL DL variant of the Web Ontology Language. We have so far found this language to be sufficient for our needs. Furthermore, it allows for a certain level of reasoning even without the use of F-logic. This, in turns, allows for the reusability of the world descriptions for other type of applications not using F-logic, but still needing some guarantees for reasoning.

On the other hand, we recognize the possible need to migrate to OWL Full at a certain stage for meta modelling of more complex worlds with extended dynamic behaviour. Our tools have thus been built with this in mind.

### 4.1 Adding Rules

To help domain expert formulate rules, we have created an intuitive user interface in OntoWorld that does not require users to be knowledgeable of F-logic. Domain experts are able to visually create rules related to concepts on a domain level. These rules are later automatically translated to Flora-2 [FLORA-2 2009] expressions using the formalized schema described in the previous section and added to the knowledge base of the reasoned engine. Complex rules can, in most cases, be broken down into simpler expressions and joined using logic expressions. However, if such approach is not possible, expert users familiar with F-logic, can formulate rules directly in Flora-2 and add them to a rule file. Figure 4 shows the user interface for adding rules in OntoWorld. It is possible to group objects together and combine method to create more powerful expressions. Objects and properties are directly extracted from the OntoWorld ontologies and pre-populated in the user interface to simplify the task of adding rules and minimize the chance of making errors.

![Image of rule adding in OntoWorld](rule_adding.png)

**Figure 4: Adding Rules in OntoWorld.**

Notice that we have talked about creating rules (or constraints) and methods. Methods are there to help expressions that are not enforced like rules, but can be used to create rules. Differentiating between rules and methods is done through the choice of parameters in an expression. For instance defining a property as “having” a certain value will automatically make the expression a
method, while choosing the parameter "must have" will make it a rule.

4.2 F-Logic and Reasoner
We have chosen to use FLORA-2 [FLORA-2 2009], which is a dialect of F-logic. Flora-2 is an advanced object-oriented knowledge base language and reasoner based on F-logic, HiLog and Transaction Logic.

4.3 Validation
The final step in our extension to the VR-WISE approach is the validating step, i.e. checking the properties and relationships present in the virtual environment against the pre-defined set of rules. As we have mentioned earlier, domain experts can graphically create rules using OntoWorld. Furthermore, it is possible to create generic rules that would hold for most VEs. For instance, we can dictate that no objects shall have negative values for properties such as height, width, etc. In the same way, we can state, for instance, that no objects should be positioned at opposite ends of a reference object unless it is surrounding it. In the same line of thought, it becomes also possible to check the semantic correctness of dynamic behaviour of objects within the virtual world. Take for example the case of a balloon that starts spinning when touched. We could define a rule that would check that a trigger (i.e. user action) is indeed defined for every dynamic behaviour and report back to the designer if one is missing.

4.4 Code Generation
Once the Conceptual Specification has been validated against these semantic rules using the reasoning engine, the code can then be automatically generated from OntoWorld. The code generated is in X3D format [Brutzman and Daly 2007] so that it can be used by different X3D players such as Vivaty [Vivaty 2009], BSContact [Bitmanagement 2009], Xj3D [Xj3D 2009].

5 RESULTS
In order to validate our approach, we will consider in this section some cases. We will take the example of a virtual city with different elements. We first start by defining constraints on specific object properties then create rules for the positioning of objects relative to other objects or reference planes. Finally, we show how different expressions can be combined to create more powerful rules.

5.1 Scenario 1: Checking for property errors
Suppose we would like to create a world where particular objects hold a specific property or a minimal or maximal value of it. If we consider again our virtual city example, we could consider forbidding creating bridges with a height of less than 2 meters. This is done in order to allow for instance, avatars and cars to pass under them. In OntoWorld, concepts are firstly being created and then instantiated like in Object-Oriented (OO) languages. The instances of these concepts are the actual objects populating the virtual world.

In order to create this rule, the domain expert must first create a bridge concept in OntoWorld and assign a default value for its height property of, say, 2 meters.

This would be translated in F-logic as follows (short form):

\[
bridge::concept[height→2].
\]

It is now possible to create a rule that checks for bridges with heights lower than 2 meters. Creating this rule in OntoWorld is done using the Property constraints section of the user interface as shown in the figure 5.

![Figure 5: The Bridge Minimum height Rule](image)

This rule is then translated in Flora-2 as:

\[
inconsistent("Inconsistent Property Value: bridges must have a minimum height of 2m") = \exists x::type→bridge, height→y, y<2]
\]

If one instance of bridge is created with a value less than the above-mentioned minimum height, the rule will be triggered and will warn the designer about this. The method \textit{inconsistent} shown above is used to return an alarm if the right hand side expression is true. This will display a message box telling the designer that a property of specific instance of bridge should be adapted in order to comply with the rules of the domain.

5.2 Scenario 2: Checking for positioning errors
In this scenario, we will consider the case of objects which should not be positioned above the ground, for instance, but should be positioned on the ground (i.e. objects positioned with a z value greater than zero if the ground reference is positioned at a value z=0). Of course, we also need to consider the case of airborne objects such as planes and birds for which this rule does not hold.

First, we start by defining a method that checks for objects that are positioned with a value superior to zero with respect to the ground. Here we use also the implicit spatial relation (see Figure 6).

![Figure 6: The Objects positioned Above Ground relations](image)

This relation is then translated in Flora-2 expressions as follows:
aboveGroundRelation(?x):- ?x[hasSpatialRelation->?r] and ?r [spatialrelationdirection->above and spatialrelationreference->‘the centre of ground’ and spatialrelationinstance->?z and ?z >0].

We also need to define a group of objects, such as airplanes, birds, etc. that are allowed to be positioned in the air. This is given in Figure 7.

![Figure 7: The Grouping of objects that are ‘airborne’](image)

The corresponding Flora-2 expression is given by

```
airborneObjects(?x) :- ?x[type->airplane or type->bird].
```

airborneObjects(?x) is defined as an object of type airplane or bird.

We are now ready to define our rule as a combination of the above two methods as given in Figure 8:

```
Inconsistent(“Inconsistent Positioning: Objects Positioned Above ground”):- abovegroundRelation(?x) and not airborneObjects(?x).
```

![Figure 8: The Rule ‘Objects Positioned Above Ground’](image)

This rule will be triggered each time the relation aboveGroundRelation(?x) is true and airborneObjects(?x) is false. In other words, instances being not an airborne object and being above the ground will be inconsistent. The reasoning engine using that rule will automatically trigger a message for such instance and will tell to the designer that these instances are not positioned correctly.

5.3 Scenario 3: Combining property and spatial relations

It is, of course, possible to combine different types of checking to improve accuracy. For instance, if we take our earlier example of minimal height for the bridge, we could consider different values based on where a bridge is located in the virtual world. Let’s suppose that the domain expert wants to dictate that a minimum height for the bridge is, for example, two meters unless a particular bridge passes above a river. In which case, this value will be modified to three meters. We can express this rule by combining property and spatial relation checking. Figure 9 gives us the different methods used to express the rule. We first start by defining the method shortBridge, which identifies all bridges with a height less than 3 meters. Next, we define the relation aboveRiver, which identifies all objects positioned above a river. Finally, we combine both methods with the logical function and to create the rule BridgeMinHeight.

![Figure 9: The Combination of property and spatial relations](image)

The corresponding Flora-2 expressions are given below:

```
shortBridge(?x) :- ?x[type->Bridge and height->?y and ?y < 3].
```

inconsistent(“Bridge height smaller than minimum allowed for bridge passing above river. bridge height must be at least 3m”):- shortBridge(?x) and aboveRiver(?x).

This rule will be triggered each time an instance of bridge has a height less than 3 meters except if the bridge is above a river.

5.4 Scene

Figure 10 shows the scene that has been generated with the approach in X3D.
6 RELATED WORK

In this section, related works will be reviewed. We concentrate on approaches aiming at designing VE applications at a higher-level of abstraction than implementation code.

The lack of high-level design methodologies for Virtual Reality (VR) has also been addressed in [Tanrriverdi and Jacob. 2001] with the presentation of VRID (Virtual Reality Interface Design).

![Figure 10. Scene generated in X3D](image)

In their paper, four key components when developing Virtual Reality (VR) interfaces are identified: object graphics, object behaviours, object interactions and object communications. The VRID methodology divides the design process into a high-level and a low-level phase and uses a set of steps to formally represent the environment. Although this methodology helps the developer to split the design into different steps and then refine them, it 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.

The Virtual Environment Development Structure (VEDS) described in [Wilson et al. 2002] is a user-centred 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 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 VEDS approach is a methodology in the strict sense of the word, i.e. guiding the designer in taking design decisions benefiting the usability of the virtual environment. There is no intention whatsoever in automatically generating the virtual environment from the high-level specifications created during the conceptual modelling phase.

The Concurrent 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 behaviour. Although the approach provides a way to design VR 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, which in our opinion is still rather close to the implementation level and not always at a conceptual level. A more detailed description of the approach together with examples can be found in [Seo et al. 2002].

The Ossa system is an approach to conceptually model of VR systems [Southey and Linders 2001]. Ossa provides a modelling 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 et al. 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 understandable 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. 2002]. 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.

7 CONCLUSIONS AND FUTURE WORK

This paper has introduced an approach that allows first to capture the semantic rules of a domain and then use them to validate the design of a related VE application. To reason about the design, all the semantic rules of a domain are translated in Frame-logic which can then be used by a reasoning engine to reason on the semantic representation of the virtual worlds and spot errors from the domain point of view for which it has been developed.

This approach has the advantage of reducing development times of virtual environments as errors and inconsistencies can be automatically spotted and quickly acted upon according to domain specific rules. This, in turns, allows the creation of more robust and conform applications. All of this contributes to better user experience and enhances the success ratio of VE applications.

Furthermore, this paper has also presented how this extension has been tested by having extended the tool called OntoWorld that implements the VR-WISE approach. This approach is scene-independent and generic enough to allow reusability.
Until now, we have only explored the possibility of checking static worlds. Future area of research can include checking for dynamic behaviour of the different objects in the world. We also recognize that the proposed user interface for adding rules to virtual environments has its limitations when real complex expressions involving multiple variables need to be created. While it is always possible to add such expressions directly to the knowledge base in the Flora-2 language, this might not be the preferred method for domain experts with little knowledge of F-logic or other programming languages. Future work could focus on simplifying this process further.

Finally, the use of an open standard based scheme for rule encoding could further help the widespread use of this approach.

REFERENCES


MAY, W. 2006. How to Write F-Logic Programs in Florida. Institut fur Informatik, Universitat Freiburg, Germany.


STUDIO MAX: http://usa.autodesk.com/


X3D: http://www.x3d.org/ accessed 2009

Xj3D: http://www.xj3d.org, accessed 2009