Structured Design of Interactive VR Applications

Krzysztof Walczak*
Department of Information Technology, Poznań University of Economics, Poland

e-mail: walczak@kti.ae.poznan.pl

Abstract

A novel approach to structured design of complex interactive virtual reality applications, called Flex-VR, is presented. Two main elements of the approach are: first, componentization of VR content, which enables to dynamically compose interactive behavior-rich virtual scenes from independent components, and second, a high-level VR content model, which enables users to easily create and manipulate complex VR application content.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism – Virtual reality; H.5.1 [Information Interfaces and Presentation]: Multimedia Information Systems – Artificial, augmented, and virtual realities

Keywords: virtual reality, 3D web, interactive 3D, behavior modeling, Flex-VR, Beh-VR

1. Introduction

One of the main problems that currently limit wide use of VR applications on everyday basis is related to the creation of complex interactive behavior-rich content. Practical VR applications require large amounts of content, which – in fact – could be, in most cases, replaced by a single command, such as Object.MoveTo(x,y,z,time). Moreover, the code is not really readable for a programmer. For example, in many cases the nodes and routes must be separated in the scene.

The remainder of this paper is organized as follows. Section 2 presents an overview of the state of the art in 3D content creation and programming. Section 3 contains an overview of the Flex-VR approach. In Section 4, the Beh-VR component model is described in details. In Section 5, the Flex-VR content model is presented. Section 6 provides an example of a Flex-VR application. Finally, Section 7 concludes the paper.

2. State of the Art

2.1 Creation of 3D content

The most important standards for describing three-dimensional content on the web are VRML/X3D [VRML, X3D] and MPEG-4 [MPEG-4]. Preparation of interactive 3D web content involves three steps: creation of 3D models and animations, assembling the models into virtual scenes and programming scenes’ behavior.

Increasing availability of automatic or semi-automatic 3D scanning tools helps in acquisition of accurate 3D models of real objects or interiors [Sitnik et al., 2005][Hengel et al. 2007]. Progress is still needed to make the process fully automatic, enable scanning of objects with arbitrary shapes and animated objects, and acquisition of advanced surface properties, but the progress in this field is very fast.

Designers can use 3D design packages such as Autodesk’s 3ds max, Maya, Softimage’s XSI or open source Blender to refine or enhance the scanned models and to create imaginary objects. The same tools can be used to assemble the 3D objects into complex virtual scenes.

Programming behavior of a virtual scene is usually the most challenging task. In VRML/X3D/MPEG-4 behavior specification is based on the dataflow paradigm. Each node can have a number of input (eventIn) and output (eventOut) fields that receive and send events. The fields can be interconnected by routes, which pass events between the nodes. For example, a touch sensor node may generate an event, which is routed to a script that implements some logic. As a result of the event processing, the script may generate another event, which may be routed to some execution unit, e.g. initiate a time sensor, which is further connected to a position interpolator, which may be connected to a transformation node containing some geometry. In such a way activation of the touch sensor may start animation of some geometry in the scene.

The dataflow programming is a powerful concept, which enables efficient implementation of interactive 3D graphics, in particular providing smooth animations, inherently supporting concurrent execution, and enabling various kinds of optimization. However, this approach has also several important disadvantages, which become more and more apparent with the shift from static 3D web content model to behavior-rich 3D web applications model [Bues et al., 2008].

First of all, programming complex interactions in this approach is a tedious task. In the example with a touch sensor given above many lines of code are required to implement a simple animation, which – in fact – could be, in most cases, replaced by a single command, such as Object.MoveTo(x,y,z,time). Moreover, the code is not really readable for a programmer. For example, in many cases the nodes and routes must be separated in the scene.
More important, however, the dataflow graph is realized as a separate, generally non-hierarchical structure, orthogonal to the main scene graph describing the composition of a 3D scene. Interweaving the two graph structures results in the increase of conceptual complexity, but also practically precludes possibility of dynamic composition based on the hierarchical organization of the 3D scene graph structure, which is critical for building dynamic 3D applications.

In many cases the scene structure must be changed (e.g., an object must be added or removed) as the result of user actions or changes in the application state. Since the routes are statically bound to the nodes they connect (by the node names), nodes cannot be simply added or removed from the scene. Every time a node is added or removed, the set of routes in the scene must be changed, depending on the node name and the set of events it supports.

Changing the set of nodes during the design phase requires either manual programming or sophisticated tools that “know” the characteristics of the scene nodes being manipulated. Changing the set of nodes in a scene during the runtime leads to complex, self-mutating code, which is difficult to develop and maintain. Supporting the change of scene contents both in the design phase and in the runtime is even more challenging.

In behavior-rich scenes major part of the scene definition is devoted to scripts implementing the scene logic and routes connecting nodes in the scene. Dynamic scenes, in which objects are added or removed in the runtime, require meta-code in scripts responsible for creation of the scene code. But what if a dynamic element, which adds or removes objects should be also dynamically added?

Creation of interactive 3D content using the above methods, requires at least a 3D designer and a programmer (which may be the same person). However, as mentioned in the introduction, practical 3D web applications require large amounts of complex interactive content, which – in most cases – must be created by domain experts, such as salesmen, teachers or museum curators. However, domain experts cannot be expected to have experience in 3D graphics design and computer programming. Therefore, the process of creation and modification of content must be much more efficient than it is possible today using the tools described above.

### 2.2 Programming behavior of 3D content

Significant research effort has been invested in the development of methods, languages and tools for programming behavior of virtual scenes. These approaches can be classified into three main groups.

The first group constitute scripting languages for describing behavior of virtual scenes. An example of a scripting language designed for creating VR interaction scenarios is MPML-VR (Multimodal Presentation Markup Language for VR) [Okazaki et al., 2002]. Another interesting example is VEML (Virtual Environment Markup Language) based on the concept of atomic simulations [Boukerche et al., 2005]. An extension to the VRML/X3D standards enabling definition of behavior of objects, called BDL, has been described in [Burrows and England, 2005]. Another approach, based on the concept of aspect oriented programming has been proposed in [Mesing and Hellmich, 2006]. Ajax3D is a recently developed method of programming interactive web 3D applications, based on combination of JavaScript and the Scene Authoring Interface (SAI) [Parisi, 2006].

The second group of solutions are integrated application design frameworks. Such frameworks usually include some complex languages and tools that extend existing standards to provide additional functionality, in particular, enabling specification of virtual scene behavior. Interesting works include Contigra [Dachstl et al., 2002] and Behavior3D [Dachstl and Rukzio, 2003], which are based on distributed standardized components that can be assembled into 3D scenes during the design phase. However, this approach is still based on the dataflow paradigm, and standard event processing making it difficult to specify more complex behaviors. A content adaptation framework based on Contigra has been described in [Dachstl et al., 2006]. Another solution, based on the use of distributed components accessible through web services has been proposed in [Zhang and Gračanin, 2007].

The common motivation for developing new scripting languages and content design frameworks, as those described above, is to simplify the process of designing complex 3D presentations. However, even most high-level scripts and content description languages become complex when they are used for preparing complicated 3D presentations. The third group of solutions try to alleviate this problem by using graphical applications for designing complex 3D content. Virtools Dev is a good example of this type of tools [Virtools]. Recent research works in this field include [Arjomandy and Smedley, 2004] and [Vitzthum, 2006]. However, even if graphical specification of behavior may be in some cases more intuitive than programming, users still must deal with complex diagrams illustrating how a scenario progresses and reacts to user interactions. Such diagrams are usually too difficult to be effectively used by non-programmers.

The approaches described above may be successfully used by 3D designers and programmers for preparing fixed 3D scenes. This, however, is not sufficient to enable widespread use of 3D applications. Such applications require creation of large amounts of complex interactive content by domain experts, such as museum curators or teachers. This requires a new method of creating interactive 3D content, which would enable this task to be performed by non-experienced users.

### 3. The Flex-VR Approach

#### 3.1 Motivation

Increasing dynamism and the central role of behavior in current 3D web applications renders the current model based on a scene graph with additional independent behavior graph impractical. Efficient creation of dynamic highly-interactive 3D applications requires a new paradigm.

First of all, we need to realize that a 3D application is much more than just the presented 3D content, in the same way as, for example, a text editor is much more than the set of graphical widgets it is currently displaying. Second, the 3D content (geometry, audio and possibly also other modalities in near future) is just a way of communicating the state of the application to a user. 3D objects may appear, disappear, and be replaced by other objects as the application state changes. Third, in case of an interactive application, it is much more likely that the presented 3D content will change, than that the way the application operates will change.

These observations are known to programmers creating 3D applications based on a scene graph APIs (such as OpenInventor,
In such applications the scene graph represents the current state of the application 3D interface, while a separate application layer is responsible for manipulating the scene graph and handling events. With VRML/X3D a similar functionality can be achieved using the EAI/SAI (External Authoring Interface/Scene Access Interface) interfaces respectively. In X3D, external SAI interface enables other applications (e.g., applets or other components on a web page) to communicate with a 3D scene. Internal SAI interface enables similar communication from within the 3D scene – using a Script node. A Script node may contain a program (or script) in any programming language supported by the browser (typically Java or ECMAScript). A script may communicate with other scene nodes by receiving and sending events through standard routes.

In VRML/X3D scripts are often used to perform some specific simple tasks in a 3D scene (e.g., open door if some condition is satisfied). However, a program contained in a Script node can be arbitrarily complex. In an extreme case, a single Script node can create and manipulate the whole 3D scene. In such case, the SAI acts as the scene graph API, while the script is equivalent to the application layer. This solution is powerful in that it enables flexible creation and manipulation of the scene, but it does not solve the problem of efficient content creation. In fact, preparing a 3D web application in such way requires even higher expertise than the classical approach.

To simplify creation of interactive content by common users (the domain experts), we need to allow the content to be built from predefined building blocks – components. Basic level of componentization is supported by the VRML/X3D/MPEG-4 standards through prototyping, which enables definition of new types of nodes based on existing types of nodes. However, this mechanism is meant strictly for programmers and therefore it is not suitable for our needs. Also, solutions enabling content creation based on purely 3D components (see Section 2.2) are not sufficient for building 3D web applications, in which behavior plays the primary role. In such applications, some components may be classical 3D components, while others should implement sounds, scenarios, means of interaction, sensors, schedulers, etc.

Clearly, there is a tradeoff between the flexibility of content creation tools and their ease of use. Generally, the more an authoring environment allows a user to do, the more difficult it is to operate. In Flex-VR, a content creator may build virtual scenes by assembling components taken from a predefined library. It is relatively easy to compose a scene, but the process is less flexible. However, additional functionality may be achieved at any time by adding new components to the library when required. This task is performed by programmers or 3D designers. This solution enables splitting the roles of domain experts, which may easily create 3D content using the available component libraries, and programmers that may develop the libraries by adding components implementing additional functionality.

3.2 Overview of Flex-VR

The Flex-VR approach targets the requirements of emerging virtual reality applications by providing means of configuration of behavior-rich interactive content from high-level components by both expert and non-expert users.

Flex-VR applications are based on configurable content, i.e., content that may be interactively or automatically configured from separate components. Interactive configuration of content enables efficient production of content by both expert and non-expert users, without going into details of 3D design or programming processes. Automatic configuration of content enables adaptation of content to various requirements such as the target environment or the target group of users.

The Flex-VR approach is based on two main elements:

- **Beh-VR content componentization**: specific organization of VR content based on a novel Beh-VR method, which enables dynamic composition of interactive behavior-rich scenes;
- **VR content model**: a high-level generic persistent VR data model suitable for dynamic composition of VR application content.

To provide a sufficient level of flexibility in dynamic configuration of complex interactive 3D content, some specific organization of the content is required. In Flex-VR, content is specifically organized following a novel method called Beh-VR. In the Beh-VR method, content is composed of independent high-level objects, called VR-Beans, that can be freely assembled into 3D scenes.

The generic Flex-VR content model describes a VR application on a higher level of abstraction than a content description language (such as VRML/X3D). Particular virtual scenes or sequences of virtual scenes are specific projections of the generic model. This generic high-level model is used to manipulate the virtual reality application, allowing this task to be performed by domain experts using simple tools without any specific knowledge in 3D application design or programming.

4. Beh-VR Component Model

4.1 VR-Beans

In the Beh-VR method a VR scene is built of software components called VR-Beans. Technically, VR-Beans are objects, implemented as standard Script nodes, but conforming to a specific convention. Conformance to this convention enables combining arbitrary sets of VR-Beans into technically correct 3D scenes and provides means of inter-object discovery and communication. Beh-VR applications are fully compliant with existing 3D content description standards and therefore can run in standard 3D browsers.

Each VR-Bean consists of at least one scenario script, an optional set of media components, and an optional set of parameters (Figure 1). The scenario script is the main element controlling each VR-Bean. Scenario scripts are programmed in a high-level XML-based programming language called VR-BML (Behavior Modeling Language). Scenarios describe what happens when the object is initialized, what actions are performed by the object and what are the responses of the object to external stimuli. In some cases there may be several different scenarios controlling the VR-Bean depending on the presentation context.

A VR-Bean can have a number of media components, which are used for representing the VR-Bean in virtual scenes. Such media components may be 3D models (X3D/VRML), images, audio and video sequences, or texts.

A VR-Bean can be also associated with a number of parameters. Parameters are characterized by a type (integer, string, etc.), a name and a value. Parameters can be read by scenario scripts and are used in determining appearance or behavior of objects.
4.2 The VR-BML language

The VR-BML language is based on XML, which is the de facto standard for creating new languages in the domain of multimedia and particularly 3D content. Most new 3D standards such as X3D, 3D XML, and COLLADA are based on XML. XML is easy to interpret, existing software may be used for parsing and processing the program code. An important feature of XML is that the structure of XML programs may be verified against formal specification of the language used.

```xml
<Scenario version="1">
  <Initialize>
    <Set name="location" value="1"/>
    <Set name="rotate" value="(true)"/>
    <Load file="bp.wrl" comp="bp" pos="0,0,0" act="false"/>
    <Scale comp="bp" value="2,2,2"/>
    <TouchSensor comp="bp" act="true" method="tsMethod()"/>
    <Activate comp="bp" act="true"/>
  </Initialize>
  <Method name="rotate" param="rotation=true">
    <Set name="rotate" value="{@rotation}"/>
  </Method>
  <Action cond="{@rotate}" time="2000" count="500">
    <Rotate comp="bp" angle="(2*PI)" axis="0,1,0" time="1000"/>
  </Action>
  <Method name="getDescription">
    <Return value="{@descr}"/>
  </Method>
  <Method name="tsMethod">
    ...
  </Method>
  ...
</Scenario>
```

Listing 1: Example of a VR-BML scenario script

The VR-BML language consists of a list of commands, a specification of program structure in form of XML Schema and expression grammar for attribute values.

There are two main types of commands in VR-BML: **structuring commands** and **instruction commands**. Structuring commands provide the necessary structure of VR-BML scripts, which enables to correctly interpret the instruction commands. Instruction commands perform some operations, such as loading an X3D model into the scene, moving a component to a specific location, and calling a method. Most commands require parameters, which may be provided as constant values or expressions containing references to variables, public values, and events generated by scene components. An example of a VR-BML scenario script is presented in Listing 1.

4.3 Structure of Beh-VR scenes

A Beh-VR scene is dynamically created by combining independent VR-Bean objects. Each VR-Bean object is controlled by a VR-BML behavior script (Figure 2). A behavior script may load any number of media components into the virtual scene, thus creating scene components, which are geometrical, aural or behavioral manifestation of the VR-Bean. The scene components may be created during the object initialization phase or later during the object lifetime. Objects may also freely change their representations at any time. A script can create and destroy scene components and can communicate with the scene components by sending and receiving events. Each script can control all scene components it has created, but has no direct influence on other scene components.

![Figure 2: A Beh-VR structure and the resulting 3D scene](image)

Since the contents of a Beh-VR scene is composed ad-hoc, communication between objects becomes a critical element. Meaningful communication requires identification of objects present in the scene, well-defined roles of objects and existence of technical means of communication.

Identification of objects is possible due to a hierarchical system of categories and a process of registration and discovery. Each object may be registered in an arbitrary number of categories. The categories also define roles of objects in the scene. An object assigned to more than one category plays several different roles in a scene.

Communication between objects is realized using two mechanisms: **public values** and **method invocation**. Public values are named public expressions that can use variables and events from a single VR-Bean. Each VR-bean can explicitly read public values, can be notified when such a value changes, and public values can be directly assigned to input events of scene components.

Method invocation can be performed on single objects, lists of objects and the whole categories. A method consists of a sequence of VR-BML commands, which may change the state of a
VR-Bean, alter its representation in the virtual scene, invoke other methods, etc. Each method may have any number of parameters. Formal specification of parameters is provided in the method declaration, while actual parameter values are set in a method call.

5. Flex-VR Content Model

To fully exploit the possibilities of configurable virtual reality applications, a high-level model of the VR application content is required. Such a model enables efficient organization, manipulation and exchange of content between applications. The Flex-VR content model describes a 3D application on a much higher level of abstraction than a typical content description language, such as VRML/X3D. The Flex-VR content model may be stored in content databases to provide persistence, high-performance data manipulation, multi-user access and transactional processing.

5.1 Presentation spaces

In the Flex-VR approach, the VR content is organized hierarchically. The hierarchy is built of presentation spaces (Figure 3). Each presentation space may have any number of sub-spaces. The depth of the hierarchy is not limited. Presentation spaces may correspond to complete virtual environments, parts of environments, or may be used merely as containers for objects (e.g., containing alternative representations of the same object). The semantics of the sub-spacing relationship depends on the super-space. For example, it may denote spatial, temporal or logical composition, alternative representations, scenario steps, etc. In some cases there may be no semantic connection of the super-space and the sub-spaces.

![Figure 3: Flex-VR content model: hierarchy of presentation spaces](image)

Presentation spaces are containers that may hold three types of elements: instances of content templates, instances of behavior templates, and instances of content objects.

5.2 Templates

Template is a parameterized program used to generate a representation of a presentation space. There are two types of templates: content templates and behavior templates. Content templates are used to generate the representation of the space and to select content objects that should be included in this representation. Simplest templates generate scenes by combining content objects. More complex templates may additionally include background elements such as a model of a room, environmental properties (e.g., a fog), etc.

![Figure 4: Flex-VR content model: a template and a template instance](image)

Each of the included content objects may contain its own behavior script. In some cases, however, it is useful to have the same (or similar) behavior shared by a number of objects. To achieve this, an instance of a behavior template, used to generate scripts implementing common object behavior may be also included in the presentation space.

![Figure 5: Flex-VR content model: content object](image)

A content object is the basic element of the Flex-VR content model. Content objects may correspond to simple 3D objects, complex objects gathering several components – either 3D components, or other media, such as images, movies and sounds – or VR-Bean objects with their own behavior specification. A content object is independent of other content objects. It is the smallest element, which is manipulated by a presentation designer.

5.3 Content objects

A content object consists of a number (zero or more) of media components and content object metadata (Figure 5). Media components are representations of the content object in various media. Examples are 3D model, image, video sequence, audio sequence, and text. More than one media component of the same type may be associated with a content object. Media components are used to represent the content object in a virtual scene. Media components may be shared by content objects. Media components
may be associated with media component metadata providing component description.

A content object instance is a content object together with values of object presentation parameters. Similarly as with template instances, content object instances can have multiple instances in the same or different presentation spaces.

5.4 Presentation domains

A common problem in the design of real-life virtual reality applications is caused by different hardware and software environments, in which the applications must run (e.g., an immersive system versus a laboratory equipped in PC computers). Also, often the applications must be targeted at different groups of users (e.g., a group of school children playing with a system while visiting a museum versus an archaeologist trying to find some details about a specific cultural object). To enable reuse of the same generic content model at different target platforms and with different presentation methods, while keeping consistency of the designed presentation structure and the content, the notion of presentation domains has been introduced. A presentation domain corresponds to a target environment or a usage scenario for the virtual reality application.

Presentation domains form a hierarchy (Figure 6). An example of a domain can be “WEB.LOCAL”. “WEB” is the name of the parent domain, while “LOCAL” is the name of the sub-domain. “ANY” is an abstract super-domain of all domains, and has no practical implementations.

In the Flex-VR content model, in each presentation space there may be a separate template for each presentation domain. For example, a space with a virtual museum exhibition may contain three sets of content templates and behavior templates, which enable presentation of the system (1) on a high-end immersive installation in the museum, (2) remotely over the Internet on a standard PC computer, and (3) locally in the museum on small portable computers in form of an interactive guide. The templates differ in the selection of the content objects, quality of the 3D models, etc.

To make the process of designing large virtual reality applications more efficient, the notion of template instance inheritance has been introduced. If a presentation space does not contain a template instance in a particular presentation domain, first a template instance in a super-domain is used – if it exists in the presentation domain. If not, a template instance from a higher-level presentation space is used – in the same domain or a super-domain.

Since inheritance concerns template instances and not only templates, all values of template parameters set in the upper-level presentation domain or the upper-level presentation space are preserved in the sub-domain or sub-space. This significantly speeds-up the design of large content sets and enables to easily maintain visual and behavioral consistency of the virtual scenes.

5.5 Presentation properties

To enable presentation designers to customize Flex-VR presentations, the content model provides a notion of presentation properties. Presentation properties form a ternary relationship between the presentation spaces, content objects and media components (Figure 7). Each presentation property is assigned to a presentation space, and it may be also optionally assigned to a content object or to a content object and a media component.

Figure 7: Flex-VR content model: presentation properties

If a presentation property is assigned to a presentation space only, it describes the presentation space. This value can be used by any template used to represent this space in a virtual scene. An example of space-level presentation property can be a description of a presentation space.

If a presentation property is assigned to a presentation space and a content object, it describes the presentation of this content object within the presentation space. A textual label describing an object is a good example of a content object-level presentation property.

If a presentation property is assigned to a presentation space, a content object and a media component, then it describes presentation of this particular media component, within the content object within the presentation space. Scale of a 3D model is an example of a media component-level presentation property.

5.6 Presentation structure

The overall structure of a Flex-VR presentation is shown in Figure 8. A presentation designer designs Flex-VR presentations by creating a hierarchy of presentation spaces and creating – in the presentation spaces – instances of content templates, instances of behavior templates and instances of content objects, and by setting values of presentation properties.

Different presentations can be achieved by the creation of template instances and content object instances derived from the same template or content object but supplied with different sets of parameter values. In some presentations, template or content object parameters that are not fixed by a content designer can be changed by end-users.
5.7 Complex structures

The Flex-VR content model enables creation of complex structures which are spatial, temporal and logical (STL – steel) arrangements of constituting elements. An example of a spatial arrangement of objects can be a virtual museum room with objects arranged next to each other in the 3D space. An example of temporal arrangement of objects can be museum showcase containing a 3D model of an artifact changing every minute to the next artifact from a given collection. An example of a logical arrangement can be a 3D model of an artifact, which changes to a higher level of details when a user approaches the showcase containing this model and back to lower-level when a user moves away from the object.

In the Flex-VR approach complex structures are modeled as presentation spaces with content controllers. Content controllers are generic content templates and behavior templates. Controllers can be used as general building blocks in many applications.

The components of complex structures can be either content objects or again presentation spaces. Therefore, arbitrarily complex structures can be created. For example, a complex three-level structure can be built by combining spatial arrangement at the highest level, time-dependent objects and levels of detail at the middle level and user privileges implementation at the lowest level (Figure 9).

The STL controllers in the Flex-VR model are a conceptual generalization of some structures available in content description languages (e.g., X3D/MPEG-4), such as Transform and Switch. However, in the content description languages the structures are functionally fixed, while in Flex-VR these structures are flexible. This means that, e.g., the spatial arrangement of objects can depend on the amount of space provided as a parameter by the upper-level model, while the temporal arrangement of objects can be automatically ordered chronologically using age information retrieved from metadata descriptions. The functions performed by an STL controller can be arbitrarily complex. Also, entirely new types of controllers can be build, for example taking into account user preferences or privileges.

5.8 Content patterns

To simplify creation of complex VR presentations consisting of objects with different roles and behavior, a presentation space may be associated with a content pattern (Figure 10). A content pattern defines a tree of content object categories, communication channels between the categories (method signatures, public values) and default implementation of methods and actions (possibly empty). Objects assigned to a category inherit methods and actions from the pattern, but can also override the default implementation with some object-specific implementation.

With the use of a content pattern, a designer may create a behavior-rich presentation by simply assigning object instances to particular categories. A single object instance may be assigned to several categories. The role and behavior of a content object in the presentation depends on the categories it is assigned to.

Each category contains objects with a specific role and behavior and therefore it implements a specific interface. In the example presented in Figure 10, the category User Interface/Out implements the display(text) method. The category interface is specified by a pattern designer (a programmer), but the assignment of particular objects to this category is performed by a presentation designer (a domain expert). There may be no objects implementing this method assigned to this category (text will not be displayed), there may be one object (e.g., a text window), but there may be also more objects (e.g., a 3D text object and a 2D banner).
Some methods have default implementations in the pattern. For example, the default rotate() method may rotate all scene components created by a content object by the same angle. Some objects may override the default implementation. For example, the Skeleton object may override this method to rotate several components around a common axis.

In the example presented in Figure 10 there are two objects assigned to the category User Interface/Out: Text window and Sounds. When an object calls a method User Interface/Out/display(text) the method is called in both objects. However, only the Text window object has a non-empty implementation of this method. The Sounds objects has no implementation, and therefore it inherits only an empty implementation from the content pattern. This solution brings some small performance overhead but frees the presentation designer from any programming tasks.

In a case when the same set of content objects is used with different patterns it may occur that there is no direct correspondence between the methods defined in the pattern and the objects. In such case, in order to enable the use of such objects, it is necessary to override the default pattern-object inheritance rule by explicit specification of which object method corresponds to the pattern method. This is achieved by the use of object presentation properties. For example, if the User Interface/Out category requires two methods: correct() and incorrect() indicating that a user has answered correctly or incorrectly, the content designer may assign to this category a sound object having only the play(sound) method. In such case, two mapping properties need to be defined:

User Interface/Out/correct() → play(correct), and
User Interface/Out/incorrect() → play(wrong)

Categories defined in a content pattern act as software component interfaces. There are dependencies between the categories, but not between particular objects, which may be assigned to the categories (Figure 11). While a pattern defines the overall structure of the presentation, the objects assigned to categories by a content designer provide the actual implementation of particular methods and actions. The assigned objects also contain specific media components such as 3D models and sounds. Therefore, entirely different presentations can be achieved by assigning different sets of objects to presentation spaces with the same content pattern.

The use of content patterns significantly simplifies the design of behavior-rich 3D applications, at the same time providing the level of flexibility that enables a user to configure the application from different elements in a meaningful way.

6. Example of a Flex-VR Application

In Figure 12 an example of an interactive 3D web application based on the Flex-VR approach is presented.

![Figure 12: Flex-VR interactive virtual museum exhibition](image)

The application implements an interactive virtual museum exhibition. A user navigating in a 3D virtual museum room has to assemble a complete armor from parts located in the 3D environment. A body skeleton is used to indicate the part of the armor that the user needs to find at a specific time.

The game is highly interactive, objects react to user interactions with animations, sounds and voice. Playing the game, a user learns about functions of particular objects by listening to a narration.

![Figure 13: Flex-VR content structure of the application presented in Figure 12](image)

The Flex-VR content structure of the application described above is presented in Figure 13. The application consists of one presentation space (Armor Game) based on a content pattern.
A content template (Moorish Hall) is assigned to this space. The template contains a 3D model of the museum room and a set of commands to create VR-Beans corresponding to content objects. The content pattern defines categories of content objects and provides default implementation of category methods. The pattern defines three categories: Objects, Scenario, and User Interface. The User Interface category has two subcategories: In and Out (see also Figure 10 and Figure 11).

A presentation designer (e.g., a curator in a museum) can create different presentations based on the same content pattern by using different content templates and by assigning different sets of content object instances to the categories, and setting their parameters and presentation properties (e.g., positions, sounds).

7. Conclusions

The Flex-VR approach presented in this paper enables building behavior-rich VR applications, in which content can be configured using independent components retrieved from a library. Configuration of application content allows users without programming skills, such as domain experts, to efficiently prepare large amounts of high-quality complex behavior-rich content. Availability of such content is a prerequisite for wider adoption of virtual reality applications in everyday life.

At the same time, new building blocks can be added to the library by experienced users – programmers and 3D designers – thus extending system capabilities. Behavior of such building blocks can be efficiently described using the proposed VR-BML language.

The Flex-VR approach has been successfully employed in the ARCO virtual museums system, which is now commercially exploited at several museums [ARCO][Walczak and Wiza, 2007]. Application of Flex-VR enables museum users (curators, exhibition designers) to setup virtual exhibitions based on predefined conceptual patterns but with varying content. This type of functionally turned out to be very effective and we have received positive feedback from the end-users.

References


