

# An Approach Facilitating 3D/VR System Development Using Behavior Design Patterns

Bram Pellens

Frederic Kleiner mann

Olga De Troyer

Vrije Universiteit Brussel

## ABSTRACT

High expectations and the increasing complexity of behavior in the domain of interactive 3D/VR applications demand for a better support at design time. Current development tools do not sufficiently provide any high-level design facilities; developers usually are forced to directly program the behavior. This paper discusses an approach that facilitates modeling behavior in interactive 3D/VR applications, at a higher level than programming. The approach is utilizing a graphical notation in combination with the concept of generative design patterns. Behavior can be specified by instantiating patterns and customizing them to a particular context. Furthermore, the approach allows experienced designers to specify new behavior patterns and make them available to others in a graphical way. By adopting this approach, the specification of behavior in interactive 3D/VR applications is elevated to a higher level of abstraction than is possible with low-level description formats like scripting languages.

**KEYWORDS:** Conceptual Modeling, Behavior Authoring, Generative Design Patterns, Graphical Notation.

**INDEX TERMS:** D.2.2 [Software Engineering] Design Tools and Techniques – Computer Aided Software Engineering (CASE), D.2.11 [Software Engineering] Software Architectures – Patterns, D2.13 [Software Engineering] Reusable Software – Reusable libraries, Reusable Models, I.3.7 [Computing Methodologies] Three-Dimensional Graphics and Realism.

## 1 INTRODUCTION

Developing interactive 3D/VR applications is not trivial. The majority of the effort in this area is about content creation and many resources are spent on this by the developing companies. In recent years, a number of authoring tools (e.g., Blender [21], 3D Studio Max [17]) and APIs (e.g., XNA [2]) have been developed to improve and facilitate the development of these applications. However, these tools only focus on the artwork, interfaces, 3D/VR environments, and so on but only few of them address the dynamic aspect of the application, i.e., the behavior of avatars and objects. As a result, the specification of behavior is often done by manually programming those using scripting languages. This is time consuming and error-prone.

In addition, many tools have their own scripting language, which force developers to learn the associated scripting language.

An observation is that hence in practice many people are trying to avoid developing long scripts by using and customizing existing scripts, either built-in scripts or scripts coming from external sources [16]. However, this way of working often leads to non-correct scripts or just wrongful behavior.

Thus, having a better mechanism in place to support the behavior development for interactive 3D/VR applications may facilitate the overall production process of interactive 3D/VR applications.

The goal of the research presented here is twofold. On one hand, we want to provide better support for the process of the behavior specifications by explicitly introducing a high-level specification phase. In this way, the behavior can be specified by designers (not necessarily programmers), which can concentrate on the requirements and discuss the design extensively with different stakeholders. On the other hand, we want to provide a mechanism that takes profit of existing knowledge and experience in specifying behaviors. Over the years, developers have come up with many predefined (parts of) solutions [27]. Being able to reuse these solutions may also reduce the time (and cost) needed for developing behaviors.

This paper reports on the work done in the context of the CoDePA project. CoDePA stands for *Conceptual Design Pattern Approach*. The aim of the project is to develop an *approach for specifying behaviors for interactive 3D/VR applications using behavior design patterns*, i.e., high-level design patterns that capture existing solutions for behavior, are used to elevate the specification of behavior to a higher level. To this end, the concept of Generative Design Patterns [15] is used for capturing patterns of behavior that often occurs in interactive 3D/VR applications.

The traditional concept of *design patterns* is already well known from the Software Engineering (SE) domain [23]. A design pattern specifies a well-defined solution to problems that often appear when designing and developing software. In SE, the design patterns are usually described by means of structured text, and the implementation of the pattern is left to the user [11]. The use of so-called *Generative Design Patterns* has the additional benefit that automatic code generation is possible. Therefore, the use of generative design patterns in our approach offers many benefits. Firstly, they allow capturing the expertise of others in a well-defined way. Secondly, they promote reuse since a pattern can be designed and implemented just once and then be used many times in different applications. Thirdly, the time of development can be reduced significantly by providing a library of patterns (or pieces of solutions) and allowing the designer to select a pattern and customize it to his needs.

The structure of this paper is as follows. Section 2 describes some background and related work for the research presented here. This is followed, in section 3, by an example illustrating the motives behind the approach. Afterwards, in section 4, the paper explains the approach and goes into more detail on each of the main aspects and features. Section 5 presents a discussion. Finally, a conclusion and future work is given in section 6.

---

Pleinlaan 2, 1050 Brussels, Belgium

[Bram.Pellens@vub.ac.be](mailto:Bram.Pellens@vub.ac.be)

[Frederic.Kleiner mann@vub.ac.be](mailto:Frederic.Kleiner mann@vub.ac.be)

[Olga.DeTroyer@vub.ac.be](mailto:Olga.DeTroyer@vub.ac.be)

## 2 BACKGROUND

This section gives related work in using Design Patterns. We particularly focus on the gaming domain since quite some work on patterns is found in this domain. Current games share many characteristics with interactive 3D/VR applications and hence the research results from game design could prove to be very useful in the design of 3D/VR applications [3]. The section also provides some previous work on modeling behavior in interactive 3D/VR applications that influenced our work.

### 2.1 Design Patterns in Games

The concept of design patterns has been used abundantly in the domain of computer games [14].

The Game Design Patterns project [1] is probably the most well known work which aims at investigating how the design of games can be facilitated through the use of design patterns. Over 200 design patterns have been developed so far, covering most aspects of game play. The most interesting ones, for our purpose, are those dealing with actions and events, interaction and goals. However, at this moment, they only use the patterns as a means of communicating the design and thus do not discuss or consider any implementation issues. Although the set of patterns is very useful from a knowledge point of view, its use is limited to specific phases in the development process of games.

A similar approach has been taken in [28] but here the patterns are more focused towards specific genre of computer games, namely Role Playing Games (RPG). They address the design decisions and options that a designer has or must take to achieve his goal. The overall intention is to educate game designers on how to properly implement the behaviors. However, the same arguments as in the previous approach apply here. The patterns need to be (re-)implemented each time they are used.

The work described in [7] proposes an object-oriented model to design the different aspects of a Virtual Environment (VE). Based on this model, a number of design patterns are presented which support the designer in solving recurrent design problems. This work also discusses implementation details. The downside is that programming knowledge is required. In our approach, an additional layer of abstraction is provided that allows the designer to use the patterns without the need to have programming knowledge.

In [9], a number of design patterns are presented with the focus on making games more accessible. Firstly, the usability patterns introduce concepts that help improving the usability and the interaction in games. Secondly, the accessibility patterns try to overcome the difficulties that people with disabilities encounter when playing games. Although the focus is less on the design aspect, some of the patterns could also be useful and can be incorporated in our approach.

Another approach is presented in [22]. In this work, a classification is made about the different components of an interactive system (such as a VE) and based on this classification, a number of interaction patterns are proposed that can be used to more easily design those different components. The patterns presented are oriented towards the interaction within VEs and less around behavior. The focus of our work is mainly on the development of behavior patterns, although some interaction patterns are also available in our pattern library.

The most interesting piece of work is described in [5] where an approach called ScriptEase is presented that allows creating behaviors using template-based behavior patterns. The designer can also create his own behavior patterns. The software tool that supports their model is completely menu-driven and all the options for the behaviors and scenarios are given in natural language, which reduces the need for programming skills although there is still some programming knowledge required in

order to come to a full understanding of the system. The patterns are parameterizable via an intuitive graphical user interface. The work presented in this paper has been influenced by the ScriptEase approach. However, our approach differs from ScriptEase in the way that the design patterns are expressed in a graphical notation.

### 2.2 High-Level Behavior Modeling

Here we review work done in the context of graphical notations and high-level specification formats to support the modeling of behavior in interactive 3D/VR applications.

The CLEVR approach presented in [24] expresses the behavior and function models through a combination of Statecharts together with Data Flow Diagrams (DFD). It is based on the assumption that the designer understands the UML notation, has some knowledge about Object-Oriented (OO) design, and knows how to translate behavior into DFDs and Statecharts.

In the Smart Objects approach [13], the behavior of objects is defined using a script language. Complex behavior is dealt with by means of templates organizing the most commonly used behaviors. However, a scripting language like this is not easy to use by untrained people.

In [25], VR applications are considered to be hybrid systems where the behavior should be specified as a combination of discrete and continuous components. It introduces *Flownets* which is a description formalism specifically designed for VR applications. However, the visual language lacks a notion for depth or modularity and therefore, the models easily become large and difficult to read.

Virtools [18] is a commercial environment, which also allows constructing object behavior graphically by combining a number of primitive behavior building blocks to form a behavior graph. The downside of the approach is the complexity with the low-level function-based mechanism, which tends to be less comprehensible.

The Behavior3D [6] approach is an XML based component architecture for behavior specification (in X3D). Here, the concept of behavior graphs is used where behavior nodes are used to encapsulate behavioral aspects of the component as well as its interface. However, constructing complex behavior still imposes a lot of background knowledge.

Another framework is described in [10] where so-called Behaviour Transition Networks (BTNs) are used, which are actually generalizations of finite state machines (FSMs), to accomplish specifying behavior. The state machine approach is very useful for discrete behaviors, but less useful for the continuous behaviors. Furthermore, it is difficult to describe behavior of objects that is somehow related to behaviors of other objects.

All the works above have their advantages and disadvantages. However, they typically all have one thing in common. When it comes to modeling more complex behavior, it becomes difficult to use them and they often require a lot of background knowledge. Our approach has some distinguishing features in comparison to the approaches reviewed above. Behavior is specified by instantiating patterns and customizing them to a particular context in an easy manner. In the aforementioned approaches behavior has to be constructed from scratch which tends to be more difficult when dealing with complex behavior.

## 3 EXAMPLE

To illustrate the use of design patterns in behavior specifications, an example is given, which is a small scenario inside a game. This example was provided by our industrial partner in the CoDePA project, being a Belgian game development company. The reoccurring behavior structures, or patterns, that are used in the

scenario are just a few of the many patterns that can be found in interactive 3D/VR applications.

The example scenario is that of a tavern (or virtual café) in a computer RPG (Role Playing Game) game. The tavern is a location where a lot of Non Player Characters (NPCs) are meeting and interact with each other, the player, and the environment. Here, we focus on specifying the behavior of the servant. She mainly performs two different kinds of behavior.

Firstly, *routine behavior* which is walking around in the tavern, taking orders from customers, fetching those orders from the bartender and delivering them to the customers. Secondly, *reaction-based behavior* which is giving a quick greeting when someone enters the tavern, or dropping everything and running away when someone starts a fight, and so on. After each reaction, she returns to her routine behavior.

Most NPCs in the game under consideration actually have a similar kind of behavior. Their behavior consists of two aspects, a routine behavior when no special events are happening and reaction-based behaviors where the NPCs are reacting to some special circumstances. After being interrupted by a reaction-based behavior, the NPC falls back to its routine behavior. This is an example of a structure (pattern) that can be re-used easily. The pattern for this construction is called the *Suspend-Resume* pattern.

Another pattern can be found in the routine behavior of the servant itself. The servant makes her tour between the different tables, serving each of them. Typically some kind of prescribed path is followed doing this manoeuvre. The exact locations of the different points on the path are specific for this application but the basic principle of following a route is general. Therefore, also for this structure, a pattern, i.e. the *PathMovement* pattern, can be used.

Furthermore, when a fight is started, the servant will perform a number of actions in a particular order. That is, (1) dropping the current order she holds, (2) playing a particular animation (e.g., fainting) and (3) afterwards running away scared. The actions themselves are of course specific to this particular scenario but the fact that one or more actions are happening in a particular sequence occurs quite often in these kinds of applications. We call this pattern the *Sequence* pattern.

#### 4 THE APPROACH

The work presented here is done in the context of a more general approach, called *VR-WISE* [4], which is developed to support the overall development of interactive 3D/VR applications. In this approach, the modeling of the object behavior is done by means of a graphical notation. We refer the reader to [19] and [20] for more details on the approach for modeling behavior in VR-WISE. In this section, we explain how the initial graphical behavior modeling approach (developed for VR-WISE) has been extended with patterns (the CoDePA framework).

To allow the designer to specify the behavior specifications using behavior patterns, and to be able to generate programming code for these specifications, a framework has been set up. The requirements established for this framework were as follows: (1) It must be possible for a pattern designer to easily construct new behavior patterns and add them to the existing collection of behavior patterns; (2) It must be possible for a behavior designer to easily search for a pattern in the existing collection of patterns; (3) It must be possible for a behavior designer to customize a design pattern to the particular context of use, e.g., it must be possible to adapt and adjust the pattern to the application; (4) Non-experienced users should be guided in the process of using (instantiating) a pattern.

Figure 1 gives an overview of the CoDePA framework. The framework basically includes two processes; the Pattern Usage Process and the Pattern Specification Process for respectively

using and creating behavior patterns. The Pattern Usage Process is part of the *CoDePA Conceptual Designer* module. The CoDePA Conceptual Designer is a diagram editor that allows creating conceptual models (specifications) of behaviors in a visual way. From the different behavior specifications that are created, the CoDePA Conceptual Designer can then generate the application specific scripting code. This code can be loaded by the actual runtime application. This application is responsible for visualizing the Virtual World, loading and executing the scripts it takes as input. The *Design Pattern Manager* module is added to our toolbox to interpret the pattern specifications created in the Pattern Specification Process, process them and make them available for its use in the CoDePA Conceptual Designer module. Furthermore, it allows to properly manage all the behavior patterns in our framework (organize them, create new ones, and so on).

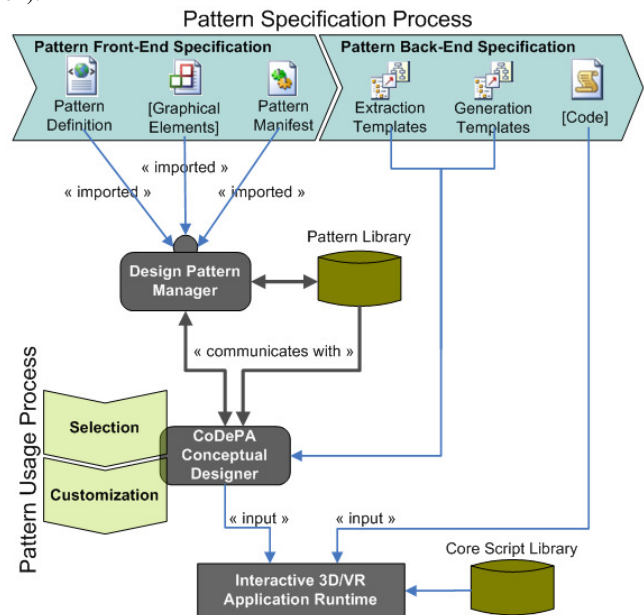


Figure 1. CoDePA framework

The following subsections describe in more detail the process of using a behavior pattern in the specification of a behavior, as well as how a new behavior pattern is created and made available in our approach.

##### 4.1 Using Behavior Patterns

Using a behavior pattern in the specification of a particular interactive 3D/VR application is called *instantiation* of the pattern. The process of instantiating a behavior pattern consists of two consecutive steps: the selection step and the customization step.

In the first step, the *selection* step, the designer selects an appropriate pattern from the collection of patterns available in our pattern library. As a pattern is expressed by means of a graphical notation, the (generic) graphical representation of the selected pattern will automatically be dropped on the drawing canvas of the CoDePA Conceptual Designer.

In the second step, the *customization* step, the designer customizes the definition of the pattern to the requirements of the behavior that he wants to specify. This includes giving proper values to parameters (overriding the default values provided in the definition), as well as adjusting the pattern to the needs of the required behavior.

We now illustrate the usage level with the modelling of the behavior described in our example, through the use of our graphical notation. First, the so-called *Suspend-Resume* pattern is selected. This pattern is used to tie the different behaviors (i.e. the routine and the reaction-based behaviors) together.

The definition of the Suspend-Resume pattern contains a number of behaviors. One of these behaviors is the main behavior; the others (minimum one) are the interrupters (i.e. interrupting behaviors). The main behavior is linked, by means of event links, to the interrupter(s). The main behavior is the default behavior that will be executed. The interrupters are those behaviors that will interrupt the default behavior. The interruption can happen when a particular event occurs and/or when a particular condition is satisfied.

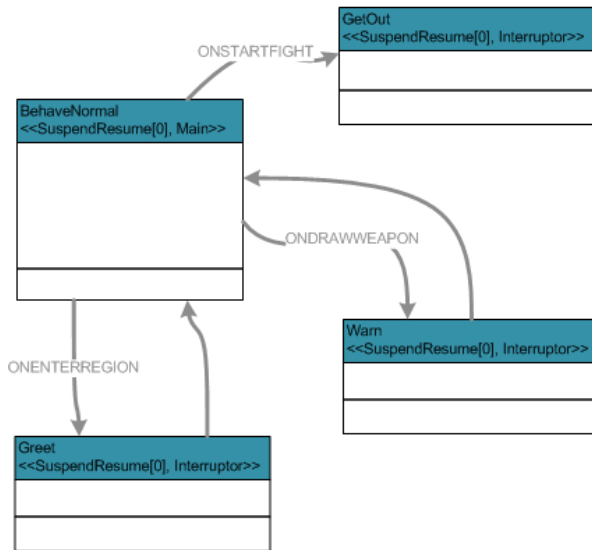


Figure 2. Graphical notation of the instantiated Suspend-Resume pattern.

Figure 2 gives the instantiation of this pattern for our particular scenario. For this pattern, there is one main behavior called *BehaveNormal*, which encapsulates a behavior which is in fact an instance of the *PathMovement* pattern. Additionally, there are a number of interrupting behaviors defined within this specification. These behaviors are *Greet*, *Warn* and *GetOut*. The interruption happens upon the occurrence of an event. The events are respectively *OnEnterRegion*, *OnDrawWeapon* and *OnStartFight*. Once the interrupting behaviors are finished, except for the *GetOut* behavior, the execution is returned back to the main behavior (because there is no return arrow from *GetOut*).

The customization is illustrated with the *GetOut* behavior, which contains an instance of the *Sequence* pattern. The Sequence pattern is a structure that occurs often in our kinds of applications. It specifies a behavior where two or more actions should happen in a particular sequence. The number and the type of actions in a sequence-behavior will vary from application to application. In the definition of the Sequence pattern, the number of actions is constrained to at least two (the minimal number to have a sequence). The pattern user can add more actions during customization. Also the pattern definition doesn't specify the exact actions. It is during the customization that the pattern user will specify the actual actions to perform. For instance, for the Flee behavior of the servant in a virtual café, three actions are needed: dropping the plate, making a (facial) animation, and running away.

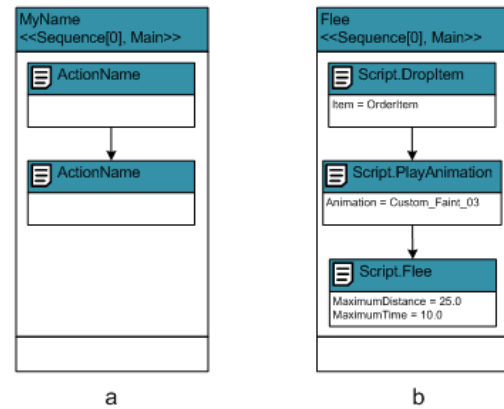


Figure 3. Graphical notation of an empty (a) and instantiated (b) Sequence pattern

So, to specify the Flee behavior of the servant, firstly the *Sequence* pattern is selected from our collection of behavior patterns. By this, the graphical representation of the pattern definition automatically drops on the drawing canvas (see Figure 3a). The main box stands for the behavior that the pattern represents. The inside of the box contains the components of the pattern that can be customized. In this case, and as explained, two, not further specified actions, are given (the two boxes labeled *ActionName*). The arrow between the boxes expresses the sequence.

Next, the designer customizes the pattern to his needs: giving proper names and parameters, adjusting the number of actions, and instantiating the actions. The result of this customization is shown in Figure 3b. We now see three actions: *Script.DropItem*, *Script.PlayAnimation*, and *Script.Flee* that are concrete actions with proper values for their parameters. Furthermore, the behavior that the pattern instantiation defines has been given the name *Flee*. The different actions in the sequence were instantiations of primitive actions available from a library in our CoDePA Conceptual Designer tool.

The remaining behaviors such as the *Greet* and *Warn* behaviors are modeled in a similar manner as the *GetOut* behavior. They are omitted and will not be discussed in detail here.

## 4.2 Creating Behavior Patterns

Obviously, in order to use a pattern, it has to be defined first. Therefore, our approach also allows building new patterns and making them available. The process of specifying new behavior patterns consists of two sequential steps, namely the *front-end specification* and the *back-end specification*. The front-end of the pattern is the part that will be visible to the user of the pattern. When using a pattern, only the front-end should be visible, while the implementation (back-end) should be encapsulated. The front-end expresses the behavior represented by the pattern at a high level of abstraction. The back-end of the pattern is the part that is needed in order to interpret the pattern and to generate scripting code.

### 4.2.1 Pattern Front-End

A first description document in the front-end that has to be provided to define a behavior pattern is the *pattern manifest*. The purpose of this document is twofold. Firstly, it specifies the metadata. The metadata describes the purpose of the pattern. This information will be used to classify the pattern in the pattern library and to allow easily searching for appropriate behavior patterns. Proper documentation is generated from this metadata

and published in different ways, i.e., in a local repository and in an online browsable library.

Important information in this description is the category to which the pattern belongs. We distinguish between two categories. The *behavior patterns* are concerned with actual actions that an object can perform in the 3D environment or VE. An example is the *PathMovement* pattern. The *structural patterns* are concerned with constructing behaviors by assembling other behaviors using some control structures. Examples are the *Suspend-Resume* pattern and the *Sequence* pattern. This last type of patterns is needed to cope with the complex issue of composing patterns [29].

Another important piece of information is the relationships of this pattern with other patterns. The following types of relationships are supported: conflicting patterns, patterns that can be instantiated by this pattern, and patterns that are changed by this pattern. This information facilitates a more advanced guidance when applying the patterns.

Next, completing the front-end of the pattern involves: (1) providing the graphical elements that are or can be used by the pattern and (2) specifying the generic behavior pattern by means of those graphical elements.

For the *graphical elements*, the pattern designer can choose from a number of predefined graphical elements. The following graphical elements are predefined: action, behavior, condition, event, and connector. An *action* represents an operation an object can perform. We provide a library of predefined actions, but also custom actions can be defined. A *behavior* represents a more complex operation than an action. *Conditions* can be used to express tests, which should be performed at runtime and which can be used to determine if and when to trigger a particular action or behavior. *Events* are used to represent interaction, either between objects or between an object and the user. *Connectors* are used to connect the actions, behaviors, and conditions in order to specify the relative order of execution.

Providing these predefined graphical elements keeps the complete set of graphical elements from becoming too large and inconsistent. However, the architecture allows adding new ones, as it is impossible to provide all the different graphical elements for all the different patterns that might be constructed in the future. New graphical elements can be created using a standard drawing tool. In this way, the graphical behavior language is extendable.

In the *pattern definition* document, the pattern designer will use this graphical language to express (in a graphical way) the generic behavior that the pattern represents. It also indicates the possible kinds of customization. Also default values must be provided for all parameters, which is done to provide as much support as possible to novice users. The format for a pattern definition is given by the Pattern Definition Grammar. It basically prescribes a pattern definition to be a kind of graph consisting of a collection of nodes, called Components, and a collection of edges, called Connections. Graphically, a component corresponds to a graphical element in the pattern and a connection corresponds to a link or connector between two or more graphical elements. Because of space limitations, we will omit the explanation of this grammar. The complete specification of the Pattern Definition Grammar encoded in an XML Schema can be found online<sup>1</sup>.

This pattern definition is captured in an XML format, which has the benefit of being tool independent. Also external tools can be used for building this description and it can be easily validated against a DTD or XML Schema. The different graphical constructs can be created using a tool like Microsoft Visio [26]. Microsoft Visio is an intuitive and easy to use drawing

application. Therefore, it does not require any specific additional skills to create new graphical elements.

#### 4.2.2 Pattern Back-End

Finally, describing the back-end of the pattern includes: (1) building a set of templates that are used to enable the generation of scripting code and (2) providing the actual implementation code that will be used by the generated scripting code.

It was decided to use a template-based code generation process [8]. Template-based code generation is a technique that enables the generation of programming code from a number of predefined templates, i.e. structures with placeholders. These placeholders are filled with the actual information from the conceptual models. This process consists of two distinct phases: *collecting metadata*, where all the necessary information from the behavior specifications is gathered; and *building and running the templates* where the gathered information is actually converted into proper scripting code (see Figure 4).

The use of code templates was chosen for two reasons. Firstly, they enhance the maintainability because what needs to be generated is separated from the actual logic that determines the generation process. Secondly, this separation between the code generator logic and the code logic itself also allows for easily generating code for different implementation platforms. The actual generation of the script code will result from the combination of the metadata (variable part) and the templates (static part). Therefore, for each pattern, a number of templates and script code need to be provided.

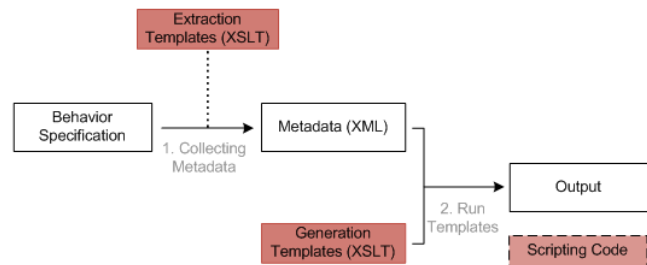


Figure 4. Generation process

Firstly, a set of *extraction templates* needs to be specified. The main purpose of these templates is to allow for the first phase of the generation process as described above (Collecting Metadata). Our behaviors are specified through a graphical notation. Internally, these diagrams are based on the Microsoft Visio file format. They also contain layout and style information, which is not needed for the generation process. Therefore, they will be preprocessed first. In other words, these extraction templates are responsible for extracting the useful information from an instantiated pattern (i.e. graphical diagram) and put it in a proper format that can be used as input for the actual generation templates (see next). Using these extraction templates, the CoDePA Conceptual Designer tool is capable of converting all possible instantiations of the pattern.

Secondly, a number of *generation templates* needs to be specified. These templates can be seen as a piece of source code containing a number of special tags referring to elements (i.e. parameters or components) given in the pattern definition. These tags (i.e. placeholders) will be replaced with the actual information given in the pattern instance. The replacement happens when the actual code needs to be generated. Then, the template gets ‘filled-in’ with the actual data from the pattern instance. This corresponds to the second phase of the code generation process as discussed above.

<sup>1</sup> <http://codepa.brampellens.be/PatternDefinition.xsd>

The last part of the back-end is the optional *scripting code* file(s), which contains code that implements the actual behavior described by the pattern. This code will be instantiated (i.e. invoked) by the output files that are generated by the code generation. The script code is optional since for simple isolated behavior, consisting of only primitive functions, the code generation can be done by using the templates only. In the case of more complex behavior, using additional functions or algorithms, the scripting code of these algorithms needs to be given. To specify the scripting code, the pattern designer has two possibilities. Either, he creates his own implementation or he reuses an already existing implementation or algorithm and uses it as a black box. In this way existing advanced algorithms can be easily incorporated. Obviously, this scripting code needs to be written in the scripting language that is used as target language for the system.

The target language that is used at the moment in our prototype implementation is the Lua scripting language [12]. This means that the templates should be written in such a way that Lua code is generated and the scripting code for the patterns developed so far should be written in Lua. However, our architecture is not restricted to this particular scripting language. If one provides another set of templates and scripting code for the patterns written in a different scripting language, then this language can be supported too.

## 5 DISCUSSION

The use of design patterns in combination with a graphical notation offers a number of advantages. (1) It simplifies the work of the designer and it improves the communication between designers as well as between designers and other stakeholders since the design patterns abstract from the implementation details. (2) The designer does not need to master a script language and also doesn't need to have good programming skills since he is making his specifications at a higher level than code. He is also completely shielded from the actual source code that will be generated. (3) Since we are dealing with generative design patterns, code can be automatically generated.

We have to note though that the creation of new behavior patterns requires designers who are more experienced in 3D/VR since, for some parts of the specifications, good programming skills are required or at least some notion of how to write the templates is needed. However, once the patterns are created and added to our framework, they can be easily used in the behavior specifications, by experienced designers as well as less experienced people in 3D/VR.

Using this approach, we have created a library containing a number of patterns from different categories. These categories include artificial intelligence (AI) patterns and interaction patterns among others. Most of the patterns in the library are behavior patterns, but also a number of structural patterns were developed, which can be used to combine either behavior or structural patterns to create more complex behavior. Note that the set of design patterns is never complete; new patterns may be discovered and can be added to the library at any time.

This approach also makes it less difficult for novice users to specify behaviors, because they can now rely on existing patterns to specify more complex behavior instead of having to specify the behaviors from scratch. As a result, it makes the design process more accessible to novice users. Taking original behavior scripts and modifying them would in this case (i.e. for novice users) often lead to a lot of errors (e.g., due to missing or confusing variables). Our approach of using the behavior patterns also allows reuse but in a controlled way.

A major drawback of diagrammatical notations is that the diagrams may become too large, especially when dealing with complex cases. By using the behavior patterns this drawback is considerably reduced as the behavior patterns act as an abstraction layer. In this way, the size and complexity of the diagrams can be decreased seriously. This makes the approach also more scalable.

## 6 CONCLUSIONS AND FUTURE WORK

In this paper, an approach has been presented to model the behavioral aspect of an interactive 3D/VR application using behavior patterns. The approach combines a graphical modeling language, which was developed earlier, with the concept of so-called visual generative design patterns.

The approach may facilitate the specification of behavior in interactive 3D/VR applications. On one hand, it allows reducing the size and complexity of the behavior specifications. On the other hand, it provides a way to take profit of existing knowledge and experience, which may also considerably reduce the time (and cost) needed for developing behaviors.

In this paper, we described how behavior patterns are created and used. For creating patterns, more experienced people are needed, but using a pattern can be done by a behavior designer, which does not need to be a skilled programmer. This is because the patterns are expressed in a graphical way and at a high level of abstraction. Implementation details are shielded from the behavior designer. The patterns can be used as first class elements in the graphical modeling language. Furthermore, as the design patterns are generative patterns, automatic code generation is provided. All the patterns that are available are put into a pattern library that can easily be searched.

Currently, the focus was on behavior patterns. However, it is our strong belief that the concept of visual generative design patterns can also be used in the context of specifying the static structure of the interactive 3D/VR application. Examples could be: patterns for objects placed according to some predefined pattern, or for rooms connected to each other in a certain way. Part of our future work will be devoted to investigate design patterns for the static aspect of these types of applications.

## ACKNOWLEDGMENTS

This research is carried out in the context of the CoDePA project (IWT070749) which is directly funded by the Institute for the Promotion of Innovation by Science and Technology in Flanders (IWT).

## REFERENCES

- [1] Bjork, S. and Holopainen, J. *Patterns in Game Design. Game Development Series*: Charles River Media, 1st edition, 2004.
- [2] Carter, C. *Microsoft XNA Unleashed: Graphics and Game Programming for Xbox 360 and Windows*: Sams, 2007.
- [3] Clarke-Wilson, C. "Applying Game Design to Virtual Environments," *Digital illusion: entertaining the future with high technology*, ACM Press/Addison-Wesley Publishing Co., pp. 229-239, 1998.
- [4] Coninx, K., De Troyer, O., Raymaekers, C. and Kleinermann, F. "VR-DeMo: a Tool-supported Approach Facilitating Flexible Development of Virtual Environments using Conceptual Modelling," *Proc. of Virtual Concept 2006*, Cancun, Mexico, 2006.
- [5] Cutumisu, M., Onuczko, C., McNaughton, M., Roy, T., Schaeffer, J., Schumacher, A., Siegel, J., Szafron, D., Waugh, K., Carbonaro, M., Duff, H. and Gillis, S. "ScriptEase: A generative/adaptive programming paradigm for game scripting," *Science of Computer Programming*, vol. 67, no. 1, pp.32-58, 2007.

- [6] Dachselt, R. and Rukzio, E. "Behavior3d: an xml-based framework for 3d graphics behavior," *Proc. of the eighth international conference on 3D Web technology*. Saint Malo, France. ACM Press. pp. 101-112, 2003.
- [7] Diaz, A. and Fernandez, A. "A pattern language for virtual environments," *Journal of Network and Computer Applications*, vol. 23, no. 3, pp. 291-309, 2000.
- [8] Dollard, K. *Code Generation in Microsoft .NET*: Apress, 2004.
- [9] Folmer, E. "Usability patterns in games," *Proc. of the Futureplay 2006 Conference*, Ontario, Canada, 2006.
- [10] Fu, D., Houlette, R. and Jensen, R. "A visual environment for rapid behavior definition," *Proc. 2003 Conference on Behavior Representation in Modeling and Simulation*, Arizona, USA, 2003.
- [11] Gamma, E., Helm, R., Johnson, R. and Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*: Addison-Wesley Professional, 1995.
- [12] Ierusalimsky, R. *Programming in Lua*: Lua.Org. 1st edition, 2003.
- [13] Kallmann, M. and Thalmann, D. "Modeling behaviors of interactive objects for real-time virtual environments," *Journal of Visual Languages & Computing*, vol. 13, no. 2, pp. 177-195, 2002.
- [14] Kreimeier, B. "The Case for Game Design Patterns," Gamasutra.com, 2002 [online], Available: [http://www.gamasutra.com/features/20020313/kreimeier\\_01.htm](http://www.gamasutra.com/features/20020313/kreimeier_01.htm)
- [15] MacDonald, S., Szafron, D., Schaeffer, J., Anvik, J., Bromling, S. and Tan, K. "Generative design patterns," *Proc. of the 17th IEEE International Conference on Automated software engineering*, Edinburgh, Scotland. pp. 23, 2002.
- [16] McNaughton, M., Cutumisu, M., Szafron, D., Schaeffer, J., Redford, J. and Parker, D. "ScriptEase: Generative Design Patterns for Computer Role-Playing Games," *Proc. 19th IEEE International Conference on Automated Software Engineering, IEEE Computer Society*, Linz, Austria, 2004, pp. 88-99.
- [17] Murdock K.L. *3DS Max 7 Bible*: Wiley Publishing Incorporated, 2005.
- [18] Nahon, D. "Virtools and virtual reality," *Proc. 2nd International Intuition 2005 Workshop*, Senlis, France, 2005.
- [19] Pellens, B., De Troyer, O., Kleinermann, F. and Bille, W. "Conceptual modeling of behavior in a virtual environment," *Special Issue of International Journal of Product and Development*. Inderscience Enterprises, vol. 4, no. 6, pp. 626-645, 2007.
- [20] Pellens, B. "A Conceptual Modeling Approach for Behavior in Virtual Environments using a Graphical Notation and Generative Design Patterns," Ph.D. dissertation, Dept. Comp. Science, Vrije Universiteit Brussel, Brussels, Belgium, 2007.
- [21] Roosendaal, T. and Selleri, S. *The Official Blender 2.3 Guide: Free 3D Creation Suite for Modeling, Animation, and Rendering*: No Starch Press, 2005.
- [22] Sanchez-Segura, M.-I., de Antonio, A. and de Amescua, A. "Interaction patterns for future interactive systems components," *Interacting with Computers*, vol. 16, pp. 331-350, 2004.
- [23] Schmidt, D.C. "Using design patterns to develop reusable object-oriented communication software," *Communications of the ACM*, vol. 38, no. 10, pp. 65-74, 1995.
- [24] Seo J. and Kim, G.J. "Design for Presence: A Structured Approach to Virtual Reality System Design," *Presence*, vol. 11, no. 4, pp. 378-403, 2002.
- [25] Smith, S. and Duke, D. "Virtual environments as hybrid systems," *Proc. 17th Eurographics Annual Conference*, N. Dodgson and M. Austen, Eds., Cambridge, UK, pp. 113-128, 1999.
- [26] Walker M., Eaton N.J. and Eaton N. *Microsoft Office Visio 2003 Inside Out*: Microsoft Press, 2003.
- [27] White, W., Koch, C., Gehrke, J., Demers, A. "Better Scripts, Better Games," *Communications of the ACM*, vol. 52, no. 3, pp. 42-47, 2008.
- [28] Whitson, J.K. "Design Patterns of Successful Role Playing Games," 2005 [online], Available: <http://legendaryquest.netfirms.com>
- [29] Yacoub, S.M. and Ammar, H. H. "Pattern-Oriented Analysis and Design: Composing Patterns to Design Software Systems," Chapter 3. Addison-Wesley. pp. 19-41, 2003.