Visual Generative Behavior Patterns to Facilitate Game Development

Bram Pellens
Vrije Universiteit Brussel
Pleinlaan 2, 1050 Brussels, Belgium
+32 2 629 37 13
Bram.Pellens@vub.ac.be

Olga De Troyer
Vrije Universiteit Brussel
Pleinlaan 2, 1050 Brussels, Belgium
+32 2 629 35 04
Olga.Detroyer@vub.ac.be

Frederic Kleinermann
Vrije Universiteit Brussel
Pleinlaan 2, 1050 Brussels, Belgium
+32 2 629 37 13
Frederic.Kleinermann@vub.ac.be

1. INTRODUCTION

The worldwide videogame and interactive entertainment industry revenue is expected to reach over $50 billion in 2009 [2]. The majority of the effort in game development revolves around content creation and many resources are spent on it. Game companies construct or buy a set of content creation tools to aid the designers in their work. However, these tools only focus on the artwork, interfaces, game levels, and so on but none of them really focus on the content which is related to the gameplay or the game story, that is, the dynamic aspect of the game. Creating this story-related content, namely the scripts for the behavioral aspect in computer games, and translating this into appropriate program code is a very difficult task. For complex scripting, the developer has to resort to manually write code using scripting languages (i.e. Lua or Python). Furthermore, these languages are not tailored for games which also do not make them easier. Over the years, game developers have come up with many predefined (parts of) solutions to improve the development process [1][3]. In addition, in practice, people are also trying to avoid writing long scripts by using existing scripts and customizing them to fit their needs.

This brings us to the concept of *design patterns*, known from the Software Engineering (SE) domain [4]. A design pattern specifies a well-defined solution to a problem that often appears when designing and developing software. Design patterns allow capturing the expertise of others in a well-defined way. In this work, we use the concept of Generative Design Patterns [5] for specifying patterns of behavior that often occur in games. The use of Generative Design Patterns has the additional benefit (over traditional design patterns) that automatic code generation is possible. In this way they promote reuse since a pattern can be designed and implemented once and be used many times in different computer games and/or in different situations. In addition, the time of development can be reduced significantly by providing a library of patterns (or pieces of a solution) allowing the developer to select a pattern and fill in the details. These patterns can be used as first class elements in our existing *visual behavior modeling language* [6] giving us a number of extra advantages. A visual language can be very helpful for conveying complex models and designs. It can help people to grasp large amounts of information more quickly than large listings of scripting code. When designed well, a visual language is easier for understanding and also for learning. Finally, the diagrams can also be used to ease the communication between different and non-technical stakeholders.

The combination of the generative design patterns and the visual language provides a powerful mechanism to model complex behaviors for computer games as the required scripting code is automatically generated from these higher-level models.

2. THE PATTERN-BASED FRAMWORK

We have developed the CoDePA framework, a pattern-based framework for specifying behaviors. This framework extends our existing behavior modeling approach, which allows specifying behaviors by means of a visual language.

The framework (see Figure 1) includes two separated processes, the *Pattern Usage Process* (lower-left on Figure 1) and the *Pattern Specification Process* (top of Figure 1) for respectively using and creating behavior design patterns.

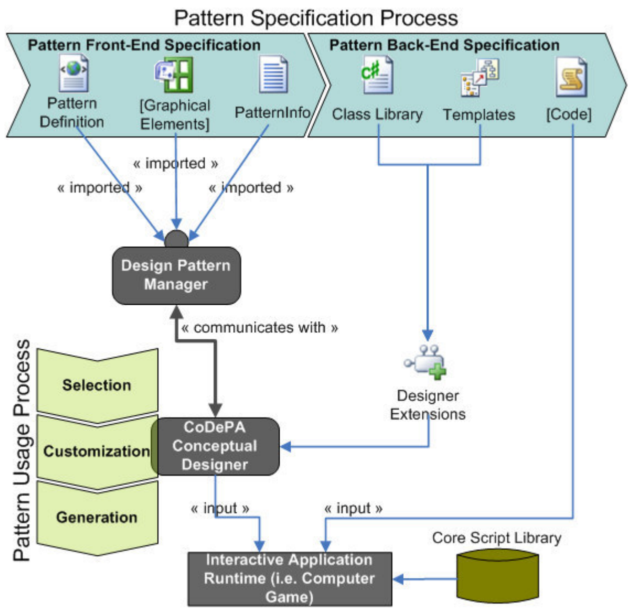

Figure 1. Pattern-based Framework

Once a pattern has been created and made available through a library, it can be applied (called instantiated) in a particular 3D interactive application (i.e. computer game). The process of instantiating a behavior pattern consists of three steps, explained below. Each step is supported by the CoDePA Conceptual Designer tool, a diagram editor supporting the approach.

In the first step, *selection*, the designer selects an appropriate pattern from the collection of patterns available in the library (and created earlier by a pattern designer). In the second step, *customization*, the designer customizes the pattern to the particular context of the game. We support two different kinds of pattern customization. A first customization is possible by giving proper values to parameters (overriding the default values provided for the pattern). A second kind of customization consists of adapting the default number of components (e.g. by adding or removing graphical elements) in the pattern. The third step, *generation*, is the automatic code generation in which all the specifications are translated into the actual scripting code.

An example of this would be a scenario in which we want to have an NPC walk according to a certain path. The first thing to do is to search for an appropriate pattern in the library. In this case, the most appropriate pattern is the *Path Movement* pattern. Selecting this pattern will automatically drop the visual representation onto the drawing canvas and the designer can start customizing the pattern to the scenario. Customizing this particular pattern means first setting a number of parameters. Amongst the parameters, the most important ones are the interpolation algorithm and the obstacle avoidance algorithm that should be used. Secondly, the actual path the character has to follow needs to be specified since this will be different for each particular scenario. Once all this information is given, the actual scripting code can be generated for the character's behavior.

Our approach also allows building new behavior patterns and adding them to a pattern library. The process of specifying a new pattern is divided into two steps, namely the *front-end specification* and the *back-end specification.* Obviously, creating a new pattern in our approach requires some knowledge of programming. However, most of the work is supported by means of the Design Pattern Manager tool.

The front-end specification focuses on the aspects of the pattern which are directly visible to the user of the pattern. As the patterns have a visual representation, the pattern designer needs to specify this visual representation, i.e. the graphical elements used in this visual representation, the roles that they play, how these elements are connected to each other, and the parameters required. This so-called *pattern definition* must be given according to an XML Schema (available online[1]). Standard *graphical elements*, with a well-defined meaning are available to ease the specification and to avoid a proliferation of graphical elements. However, new ones can be easily introduced. Furthermore, the pattern should be described using meta-data. This meta-data contains information that can assist the designer on how or when the use the behavior patterns.

The back-end specification is required in order to be able to generate scripting code. A first set of files to be given is the *class library*, which should contain code to enable the interpretation of instantiations of the newly specified pattern by the system. Also some *templates* need to be given. The templates are used to generate the actual scripting code. The data that will be given by the pattern user when instantiating and customizing a particular pattern together with the templates will be transformed into appropriate scripting code by the code generator. All specifications in our approach are based on XML, and XSL transformations are used for the generation. Finally, some scripting *code* can be given which contains the actual implementation of the behavior described by the pattern or some helper functions. It is this code that is instantiated (i.e. invoked) or used by the scripting code that was generated. In the case of simple behavior patterns which are only using primitive actions, it is not required to provide any scripting code.

3. CONCLUSIONS

The CoDePA project is elaborated in cooperation with a Belgian game development company. It aims at facilitating the game development process by means of providing generative behavior patterns. The patterns are integrated into an already existing visual behavior modeling language. Also a mechanism to specify new behavior patterns is provided.

The behavior design pattern framework, implementing the proposed approach, has been discussed in this abstract. It allows a more experienced user to create new behavior patterns and to include them in the library of the framework for further use. The patterns can be used inside the visual behavior modeling language, developed earlier to specify behavior. The design patterns created with this framework are generative, which means that automatic scripting code generation for interactive applications such as computer games is supported.

We have already applied our approach to numerous cases coming from our partner in the project in order to show the feasibility. However, in the future, we will also conduct experiments to evaluate the usability of the approach. Furthermore, we also plan to extend our pattern library with more patterns, especially towards smart behavior.

4. REFERENCES

[1] Bjork, S. and Holopainen, J. 2004 *Patterns in Game Design*. Game Development Series, Charles River Media, 1st ed.

[2] DFC Intelligence 2008 World Wide Market Forecasts for Video Game and Interactive Entertainment Industry Report Series, from http://www.dfcint.com

[3] Folmer, E. 2006 Usability Patterns in Games, in Proc. of Future Play 2006 Conference, Ontario, Canada.

[4] Gamma, E., Helm, R., Johnson, R., Vlissides, J. 1995 *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional.

[5] MacDonald, S., Szafron, D., Schaeffer, J., Anvik, J., Bromling, S., Tan, K. 2002 Generative Design Patterns, in *Proc. 17th IEEE International Conference on Automated software engineering,* Edinburgh, Scotland, 23-34.

[6] Pellens, B., De Troyer, O., Kleinermann, F., Bille, W. 2007 Conceptual modeling of behavior in a virtual environment, in *Special Issue of International Journal of Product and Development*, Inderscience Enterprises, 4(6):626-645.

[1] http://codepa.brampellens.be/PatternDefinition.xsd