Statechart modelling of interactive gesture-based applications

Romuald Deshayes
Institut d'Informatique
University of Mons
Place du Parc 20, 7000 Mons
Belgium
romuald.deshayes@umons.ac.be

Tom Mens
Institut d'Informatique
University of Mons
Place du Parc 20, 7000 Mons
Belgium
tom.mens@umons.ac.be

ABSTRACT

Developing intuitive interactive applications that are easy to maintain by developers is quite challenging, due to the complexity and the many technical aspects involved in such applications. In this article, we tackle the problem in two complementary ways. First, we propose a gestural interface to improve the user experience when interacting with applications that require the manipulation of 3D graphical scenes. Second, we reduce the complexity of developing such applications by modeling their executable behaviour using statecharts. We validate our approach by creating a modular and extensible Java framework for the development of interactive gesture-based applications. We developed a proof-of-concept application using this framework, that allows the user to construct and manipulate 3D scenes in OpenGL by using hand gestures only. These hand gestures are captured by the Kinect sensor, and translated into events and actions that are interpreted and executed by communicating statecharts that model the main behaviour of the interactive application.

Author Keywords

software modeling, human-machine interaction, natural interface, gestural interaction, statechart, 3D, Kinect, OpenGL

INTRODUCTION

Developing interactive systems can be quite complex, due to the non-trivial behaviour involved in such applications, their need to instantly react to user events, and the rising popularity of novel natural interfaces such as haptic, tactile and gestural interfaces [12, 13, 16]. The technical mechanisms underlying these systems evolve very rapidly. Hence, there is an urgent need to provide *application frameworks* to allow developing such applications in a modular and extensible way, by hiding the accidental technical complexity [4].

Model-driven engineering [17] is advocated as a discipline that allows to raise the level of abstraction of complex applications, by specifying the executable behaviour through visual models. One of the most frequently used models enabling the visual specification of executable reactive behaviour is *statecharts* [9]. Different statecharts can communicate asynchronously by sending and receiving events. Within each statechart, state changes and specific actions are realised by responding to received events. Kienzle *et al.* [11] illustrate how to use statecharts to model and implement the behaviour of game characters in computer games. Dubé *et al.* [7] explain how to use statecharts to model complex (scoped) user interfaces, focusing on their reactive behaviour.

For certain types of highly interactive applications, such as those involving 3D manipulation, traditional input devices (such as mouse, keyboard or touch screen) are suboptimal, and need to be replaced by more advanced ways to capture user input. Gesture-based input (such as hand and head movements) is likely to improve user interaction with applications requiring the manipulation of 3D graphical scenes. Typical examples of this are computer-aided design tools such as interior design software, that allow professionals and end-users to create interior home design plans by manipulating 3D objects. Using gesture-based interfaces would therefore provide a more intuitive and improved user experience.

Summing all up, the contribution of this article will be manifold. We will present a modular and extensible framework that we have developed to create highly interactive and reactive applications. The framework enables the use of novel input devices for natural interfaces. The interactive executable behaviour is specified at a high level of abstraction, using statechart models. We validated our framework by developing a proof-of-concept application to create and manipulate 3D objects. It uses a gestural interface to control objects using hand movements only, using the Kinect controller as input device.

PROOF-OF-CONCEPT APPLICATION

The application that we have chosen to develop as a proof-of-concept is a simple 3D drawing tool that enables the creation and manipulation of 3D objects using the OpenGL graphical library. Rather than using a keyboard, mouse, remote control or other traditional input device, the application is controlled by hand gestures.

Proceedings of the *First International Workshop on Combining Design and Engineering of Interactive Systems through Models and Tools (ComDeisMoto), organized at INTERACT 2011 – 13th IFIP TC13 Conference on Human-Computer Interaction, Lisbon, Portugal, September 6, 2011.*

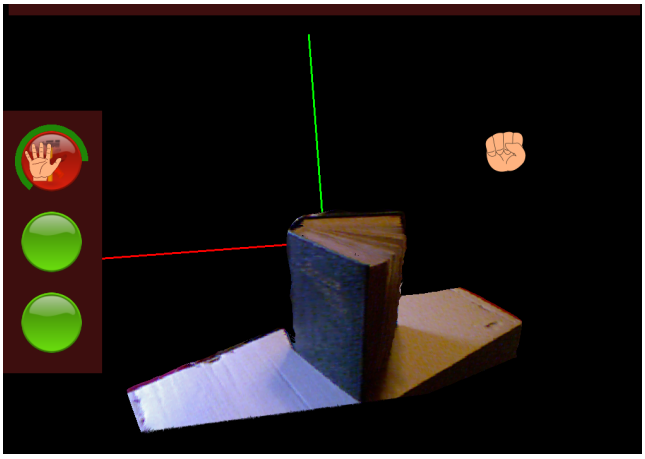

Figure 1. Screenshot of an interactive application to manipulate 3D objects using hand gestures.

Figure 1 shows a screenshot of our tool. A video of the application in action can be found on YouTube[1]. The user controls the application using his hands only. A closed right hand is shown on the right of the screen. An opened left hand is positioned on the left side of the screen, triggering the selection of the top menu button (displayed in red). In the middle of the screen, a 3D model is displayed which has been reconstructed from a real book [6]. Alternatively, the user can draw 3D models from scratch using hand gestures. Any 3D model can be rotated, enlarged and moved using hand movements.

To realise the application, we have used Microsoft's Kinect[2] as input device. This controller can detect the shape and 3D position of a person in front of the device through the use of an infrared projector, and infrared camera and a colour camera. Once the system is calibrated and a person is detected, the 3D position and movement of the hands can be tracked in real time.

In the remainder of this paper we will explain how we have realised this application by customising a generic framework we have developed for creating interactive applications that use novel natural interfaces. An important part of the framework is the ability to model the executable behaviour of the user interaction at a high level of abstraction.

MODELING GESTURE-BASED BEHAVIOUR

To specify the complex behaviour of reactive event-driven systems, of which human-computer interaction (HCI) applications are a particular instance, we wish to use visual models at a sufficiently high level of abstraction, while still maintaining the ability to execute the models. Two types of models appear to be suitable for this purpose: Petri nets [15] and statecharts [9]. Both have been suggested by other researchers for the purpose of modeling user interface behaviour [3] [7]. We have opted for statecharts for several reasons. First, Petri nets are intrinsically nondeterministic in nature, giving rise to additional problems. Second, existing tool support for executable statechart modeling appears to be more advanced. Third, the statechart notation offers many mechanisms to reduce redundancy and complexity (such as composite states, concurrent states, history states). Although similar extensions of the Petri net formalism have been proposed (e.g., hierarchical Petri nets), they do not appear to have gained widespread use.

Therefore, we believe that statecharts provide the most appropriate formalism, at the right level of abstraction, to represent event-driven, gesture-based behaviour. We are not aware of any other visual modeling approach that has been used to represent the executable behaviour of gesture-based interactive applications. Of course, many other user interface modeling approaches exist, such as ConcurTaskTrees, storyboards, UsiXML [10]. These approaches appear to be complementary to ours, as they are typically more process or task-oriented, and much less focused on the executable interactive behaviour.

THE FRAMEWORK

We have created a generic multi-threaded object-oriented application framework in Java that allows for the development of interactive applications in a modular and extensible way. Its module structure is represented in Figure 2. Figure 3 shows the UML class diagram of the framework, illustrating the main classes used, as well as their composition and inheritance relations.

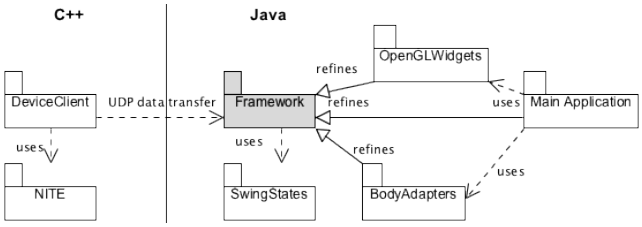

Figure 2. Module structure of the application framework.

The framework relies on an external DeviceClient module that transfers in a continuous fashion the data received from the input device in a simple text-based format to the framework. We implemented a client-server protocol to achieve a high degree of decoupling, in order to be able to accommodate a wide variety of input devices for natural interfaces. The framework provides a *Server* class (see Figure 3) that receives data from the device client over an UDP socket on a local port.

Any input device can be supported, as long as the device client respects the conventions and the format imposed by the framework. For our particular gesture-based application, the input device will be a Kinect sensor. We have implemented a device client in C++ to record at regular time intervals the data sent by *NITE*, a C++ development framework that has been provided by PrimeSense[3], the company that developed the Kinect sensor. The *NITE* framework facili-

[1] www.youtube.com/watch?v=lVcqzWTnpMY
[2] www.xbox.com/kinect
[3] www.primesense.com

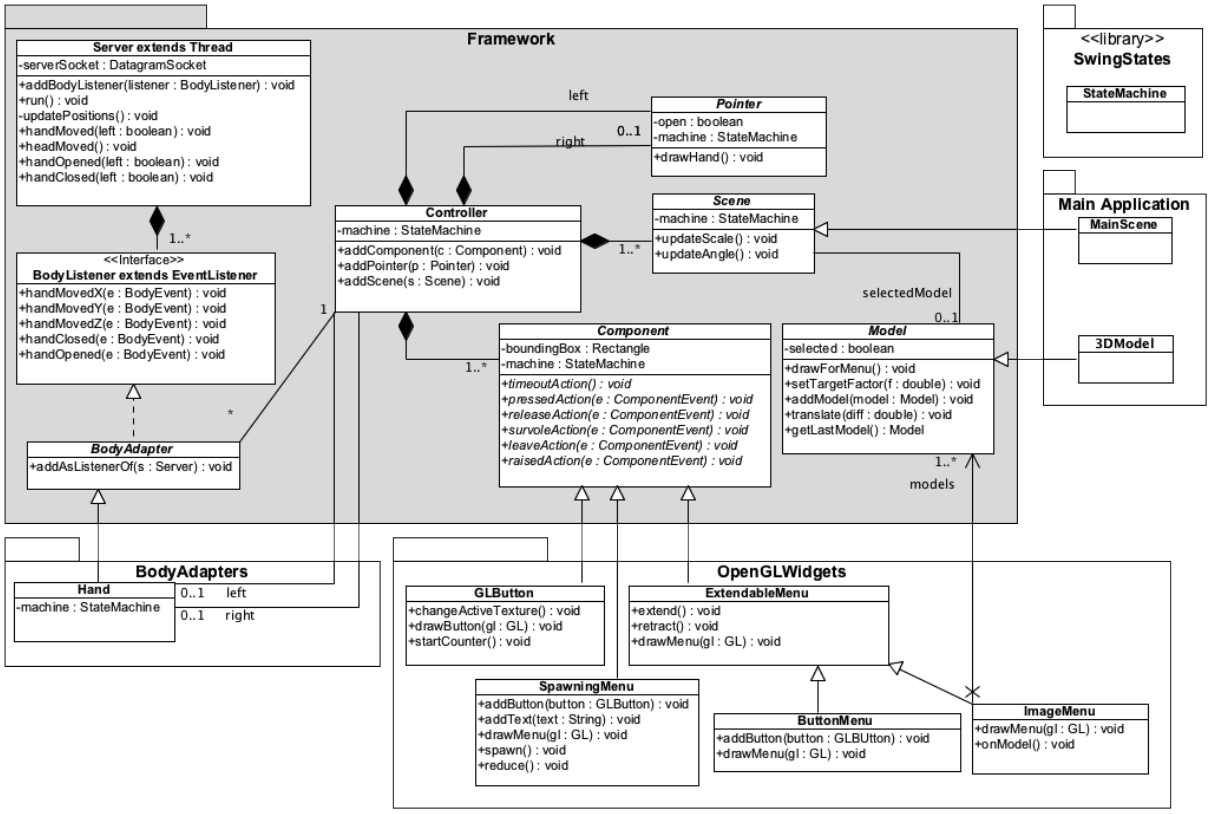

Figure 3. Class diagram of the application framework.

tates the development of interactive applications based on this sensor, by providing advanced functionality to track the user's shape in real time.

To represent graphical widgets on the screen, the framework provides an abstract class *Component* (see Figure 3). The OpenGLWidgets module contains subclasses of *Component* to represent graphical widgets (such as menus and buttons) using the OpenGL graphical library[4]. The framework also provides an abstract class *BodyAdapter* representing a particular body part that listens to events corresponding to this body part. The BodyAdapters module provides concrete classes that specialise *BodyAdapter* with the different body parts that can be used to interact with the application. Currently, we only provide hand gestures (implemented by the concrete subclass *Hand*), but head and leg movements (for example) could be added in a straightforward way.

Based on an observer design pattern (using the Java EventListener interface), the *Hand* body adapter (see Figure 3) listens to the events received from the *Server* to compute the relative positions of both hands of a person relative to his head, as well as the hand movements (based on previous positions) and speed, and the hand's status (open or closed). Since we have opted for an event-based solution, this information is transformed into specific events that correspond to hand movements: *handMovedX*, *handMovedY*, *handMovedZ*, *handOpened* and *handClosed*. Each of these events takes a boolean parameter to enable the distinction between the left and right hand, so that both hands can be used separately to control the application. Nontrivial behaviour can be implemented by triggering specific actions in response to the above events.

Since we wish to model the executable behaviour of user interaction with statecharts, our framework uses *SwingStates*[5], an efficient open source library that extends the Java Swing user interface toolkit with support for state machines [1, 2] to specify the behavior of interactive systems. While the library originally uses standard input devices (such as mouse and keyboard), its hierarchical design supports arbitrary input devices such as the Kinect to implement novel natural interaction techniques. The overall complexity of modeling the interactive behaviour is reduced by decoupling the behaviour of interconnected classes: each class provides a different statechart specifying its behaviour. These statecharts communicate asynchronously by sending and receiving events.

An interactive application based on natural interfaces can be realised by using the framework and its modules, and by specialising some of its abstract classes (such as *Component*, *BodyAdapter*, *Model* and *Scene*).

STATECHART MODELS

[4] www.opengl.org

[5] swingstates.sourceforge.net

In this section we present the statecharts that represent the dynamic behaviour of our interactive gesture-based application. Each of these statecharts is associated to one of the framework classes or a subclass thereof.

Figure 4 shows the statechart of the *Hand* class, the body adapter representing the behaviour corresponding to hand movements. It is composed of two orthogonal statecharts, one detecting the opening or closing of a hand, the other detecting the vertical position of the hand. Since two instances *left* and *right* of class *Hand* are associated to the *Controller* class of the framework, the behaviour of each hand can be controlled independently. Opening, closing, raising or lowering a hand will trigger a state change and send an event, that can be exploited to change the behaviour of the application.

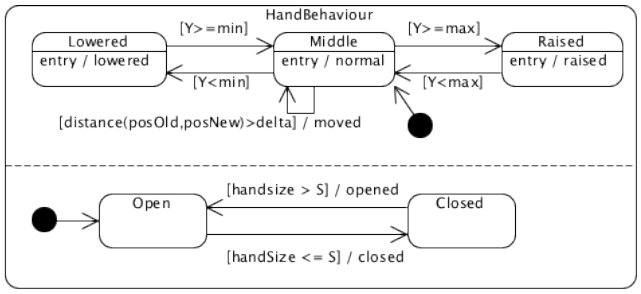

Figure 4. Statechart representing the behaviour of a hand.

Figure 5 shows the statechart that represents the behaviour of the *Controller* class. It detects, among others, which hand is closed or opened. It listens to the events triggered by the statecharts associated to the left and right hand (see Figure 4). Each event passes along a parameter *BodyEvent* that carries the information about the position of the hand, its speed and status, and the transition that was followed.

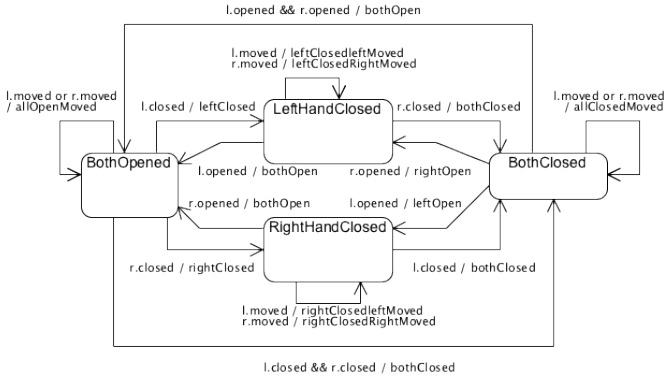

Figure 5. Controller statechart listening to both hands (left and right) to respond to movement with no, one or both hands opened.

The statechart of Figure 6 specifies the behaviour of the abstract class *Component* from which all graphical components (such as buttons and menus) that constitute our interactive application inherit. Each graphical component listens to events sent by the Controller statechart of Figure 5. The actions triggered upon the receipt of a particular event can be different for each type of component. When a hand is moved over the component, its state becomes *FlownOver*. When the hand stays in this position for more than two seconds, a *timeoutAction* will be triggered. If the hand closes when being on top of a component, the state changes to *Pressed*, and the *pressedAction* is triggered. When the hand is opened again, the *releasedAction* is triggered and the component passes to the *FlownOver* state again. This type of behaviour is for example very useful to implement drag and drop functionality to move objects around, or resize functionality.

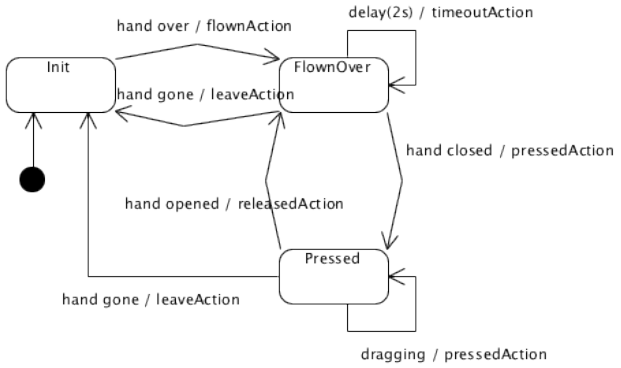

Figure 6. Statechart representing the behaviour of a graphical component (e.g., a button or a menu).

The statechart of Figure 7 represents the behaviour of the application-specific class *MainScene* that inherits from the abstract framework class *Scene*. It represents a graphical scene consisting of one or more models (such as the 3D book model of Figure 1).

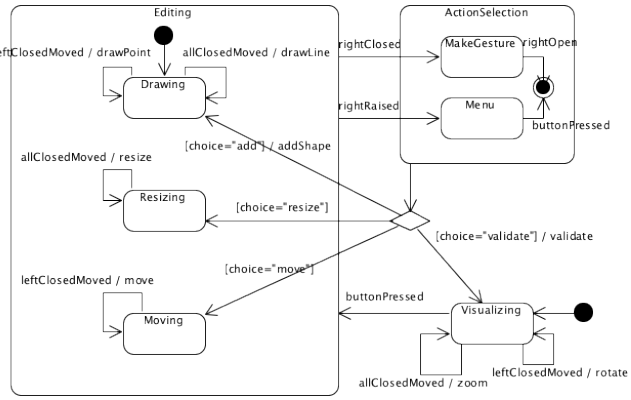

Figure 7. Statechart representing the behaviour of a graphical scene.

A graphical scene is initially in the state *Visualizing*, where the currently selected model can be zoomed or rotated with the hands. The user can select a button to carry out more specific model editing behaviour in one of the substates of the composite state *Editing*. In the *Drawing* substate, closed hand gestures are directly interpreted to draw shapes on the screen just as if the user would hold a pencil. In the *Resizing* substate, the model can be resized when both hands are closed. In the *Moving* substate, the user can move models around when keeping the left hand closed. By raising the right hand in any of these substates, a menu is spawned from

which the user can choose between the actions of visualising, drawing, resizing or moving models displayed on the main scene. Alternatively, by closing the right hand only, we arrive in the *MakeGesture* state in which the user can apply predefined gestural patterns that will, when executed, trigger one of these actions and their corresponding state changes. For example, the gesture of drawing a cross using the closed right hand will trigger a transition to the *Resizing* state. The ability to specify and detect gestural patterns and to associate them to specific actions is a feature supported by classifiers implemented in the *SwingStates* library.

DISCUSSION AND FUTURE WORK

We customised our framework into a proof-of-concept application for manipulating 3D objects using hand gestures only. We found the application to be very intuitive in use, and we did not encounter any performance problems. Nevertheless, both the framework and the application can still be improved in many ways. The framework needs to be ported to different operating systems since currently it has only been developed for a Linux OS. The client device that communicates with the Kinect sensor suffers from a number of child deseases that need to be overcome. For example, when the two hands of the user are very close to one another, the application cannot distinguish between both.

To validate and assess the usefulness of our framework, we need to carry out different kinds of controlled experiments. With end-users we can assess the improved user experience of gesture-based interfaces as opposed to traditional ones. With developers of HCI applications we can assess the extensibility and modularity of our framework. With maintainers of HCI applications we can assess the difficulty of specifying and evolving interactive behaviour that is specified by communicating statechart models.

From the modeling point of view, the statecharts are currently hardcoded in the Java framework using the *SwingStates* library. In fact, all statecharts presented in this article are flattened into state machines. We did not observe a scalability problem due to state explosion, thanks to the ability to run multiple communicating state machines concurrently. From the visualisation point of view, *SwingStates* only provides a simple state machine visualiser. It would be more useful to allow the user to specify the state-based interactive behaviour separately using some integrated visual modeling tool. From this specification, it is relatively straightforward to generate the corresponding Java code. A more advanced possibility would be to develop an interactive gesture-based application to modify the statechart behaviour at runtime.

We also need to compare, and possibly combine, our statechart models with other modeling approaches that have been proposed for user interface modeling [10] and behavioural modeling of interactive graphical applications (e.g., [18]). Similarly, we need to compare our framework with other frameworks that have been proposed for developing interactive applications based on novel user interfaces such as the (outdated) Amulet environment [14] and its successors.

In the future, we will continue to use our framework for other HCI applications, such as the algorithmic reconstruction and manipulation of 3D objects and scenes [6]. We will also validate and extend our framework for other types of natural interfaces (such as haptic/tactile interfaces), other input devices (e.g., multi-touch devices, electronic gloves), more output devices (e.g. 3D virtual reality goggles), and other body adapters (e.g., head and feet). This may lead to a significantly more interactive and more immersive user experience. For example, the combination of headtracking functionality [5, 8] (taking into account the position and the direction of the head) and 3D goggles will give the user the impression of looking into a 3D box of which he can observe and manipulate the contents from different angles.

CONCLUSION

In this article, we presented a modular and extensible object-oriented Java application framework to facilitate the development of interactive applications based on natural user interfaces. The framework is decoupled from the input device through the use of a client-server architecture. One of the advantages of the framework is that it allows to improve the user experience by enabling novel natural interfaces (such as gestural interfaces), in particular for applications that require the manipulation of 3D models and scenes.

A proof-of concept application was developed by using the Kinect sensor as input device, and by translating its input into a series of events corresponding to particular hand movements. These events are exploited by statecharts that represent the executable behaviour of the interactive application. Different communicated statecharts are associated to different framework classes, making the behaviour of the classes loosely coupled: they follow an observer design pattern and their statecharts communicate asynchronously through the sending and receiving of events.

We believe that the use of statechart models reduces the accidental complexity by raising the level of abstraction to the appropriate level, facilitating the specification of complex interactive behaviour in an intuitive visual way.

ACKNOWLEDGMENTS

This work has been carried out in the context of the masters thesis of the first author. We express our gratitude to the MINT team of University of Lille 1 for proposing this research topic. The research is co-funded by the European Regional Development Fund (ERDF), Wallonia (Belgium), FRFC project 2.4515.09 financed by F.R.S.-FNRS, and ARC project AUWB- 08/12-UMH, financed by the Ministère de la Communauté française - Direction générale de l'Enseignement non obligatoire et de la Recherche scientifique, Belgium.

REFERENCES

1. C. Appert and M. Beaudouin-Lafon. SMCanvas : augmenter la boîte à outils Java Swing pour prototyper des techniques d'interaction avancées. In *Proc. 18ème conférence francophone sur l'Interaction Homme-Machine (IHM 2006)*, pages 99–106. ACM Press, April 2006.

2. C. Appert and M. Beaudouin-Lafon. Swingstates: Adding state machines to java and the swing toolkit. *Software Practice and Experience*, 38(11):1149–1182, September 2008.

3. R. Bastide and P. A. Palanque. Petri net objects for the design, validation and prototyping of user-driven interfaces. In D. Diaper, D. J. Gilmore, G. Cockton, and B. Shackel, editors, *INTERACT*, pages 625–631. North-Holland, 1990.

4. F. P. Brooks. *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley, 20th anniversary edition, 1995.

5. R. Deshayes. Réalité virtuelle appliquée au headtracking et au handtracking. Technical report, Faculty of Science, University of Mons, Belgium, June 2010. Masters Project.

6. R. Deshayes. Reconstruction algorithmique d'objets 3D. Master's thesis, Faculty of Science, University of Mons, Belgium, June 2011.

7. D. Dubé, J. Beard, and H. Vangheluwe. Rapid development of scoped user interfaces. In J. Jacko, editor, *Human Computer Interaction (HCI 2009)*, volume 5610 of *Lecture Notes in Computer Science*, pages 816–825. Springer, 2009.

8. Y. Fu and T. S. Huang. hmouse: Head tracking driven virtual computer mouse. *Applications of Computer Vision, IEEE Workshop on*, page 30, 2007.

9. D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.

10. H. Hussmann, G. Meixner, and D. Zuehlke. *Model-Driven Development of Advanced User Interfaces*. Springer, 2011.

11. J. Kienzle, A. Denault, and H. Vangheluwe. Model-based design of computer-controlled game character behavior. In G. Engels, B. Opdyke, D. C. Schmidt, and F. Weil, editors, *MoDELS*, volume 4735 of *Lecture Notes in Computer Science*, pages 650–665. Springer, 2007.

12. T. Miller and R. Zeleznik. The design of 3d haptic widgets. In *Proceedings of 1999 Symposium on Interactive 3D Graphics*. ACM Press, 1999.

13. T. Moeslund, M. Störring, and E. Granum. A natural interface to a virtual environment through computer vision-estimated pointing gestures. In I. Wachsmuth and T. Sowa, editors, *Gesture and Sign Language in Human-Computer Interaction*, volume 2298 of *Lecture Notes in Computer Science*, pages 239–250. Springer, 2002.

14. B. A. Myers, R. G. McDaniel, R. C. Miller, A. S. Ferrency, A. Faulring, B. D. Kyle, A. Mickish, A. Klimovitski, and P. Doane. The amulet environment: New models for effective user interface software development. *IEEE Transactions on Software Engineering*, 23(6):347–365, June 1997.

15. W. Reisig. *Petri Nets: An Introduction*, volume 4 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1985.

16. P. Reuter, G. Riviere, N. Couture, S. Mahut, and L. Espinasse. ArcheoTUI – driving virtual reassemblies with tangible 3d interaction. *J. Comput. Cult. Herit.*, 3:1–13, October 2010.

17. T. Stahl and M. Völter. *Model Driven Software Development: Technology, Engineering, Management*. Wiley, 2006.

18. A. Vitzthum. SSIML/Behaviour: Designing behaviour and animation of graphical objects in virtual reality and multimedia applications. *Int'l Symp. Multimedia*, pages 159–167, 2005.