High-Fidelity Prototyping of Interactive Systems Can Be Formal Too

Philippe Palanque, Jean-François Ladry, David Navarre, and Eric Barboni

IHCS-IRIT, Université Paul Sabatier – Toulouse 3, France
{ladry,palanque,navarre,barboni}@irit.fr

Abstract. The design of safety critical systems calls for advanced software engineering models, methods and tools in order to meet the safety requirements that will avoid putting human life at stake. When the safety critical system encompasses a substantial interactive component, the same level of confidence is required towards the human-computer interface. Conventional empirical or semi-formal techniques, although very fruitful, do not provide sufficient insight on the reliability of the human-system cooperation, and offer no easy way to, for example, quantitatively compare two design options. The aim of this paper is to present a method, with supporting tools and techniques, for engineering the design and development of usable user interfaces for safety-critical applications. More precisely we present the Petshop environment which is a Petri net based tool for the design specification, prototyping and validation of interactive software. In this environment models of the interactive application can be interactively modified and executed. This is used to support prototyping phases (when the models and the interactive application evolve significantly to meet late user requirements for instance) as well as in the operation phase (after the system is deployed). The use of the description technique (the ICO formalism) supported by PetShop is presented on a multimodal ground segment application for satellite control and more precisely how prototyping can be performed at the various levels of the architecture of interactive systems.

Keywords: Model-based approaches, formal description techniques, interactive prototyping, reliability, evolvability.

1 Introduction and Related Work

Current research in Interactive systems promotes the development of new interaction and visualization techniques in order to increase the bandwidth between the users and the systems. Such an increase in bandwidth can have a significant impact on efficiency (for instance the number of commands triggered by the users within a given amount of time) and also on error-rate [23] and complexity. To address design issues raised by such systems, new design and development processes have to be defined and assessed.

Current development processes both in the field of Human-Computer Interaction (HCI) [11] and Software Engineering (SE) [9, 4] promote iteration-centered processes but with a different perspective. In the field of HCI, the product of each iteration is tested with potential users of the system under development while, in SE, the product is evaluated by different stakeholders including client or customer (the one who pays for or buys the product) and more unlikely users (but user-centered approaches (such as task analysis and modelling).

At design stage, HCI approaches promote iteration through the production of prototypes to be presented to and used by "real" users. While such design process is widely agreed upon, the debate is still vivid whether one should use low-fidelity [24] or high-fidelity prototyping [26, 14].

When it comes to complex applications at the interaction level [19], or at the application level [25], low fidelity approaches only address a small part of that complexity. The outcome is too informal for making it exploitable further on in the development process without losing a significant part of it. This limits the use of low-fidelity prototyping approaches to the earlier phases of the development process, where main design questions are addressed and low level ones left to later phases.

The main drawback of high-fidelity prototyping lays in the fact that the iterations are more time consuming and thus prevent exploration of new ideas without jeopardizing the entire project by overrun on schedule. Another inconvenient of high-fidelity prototyping is related to the product of that phase which most of the time corresponds to program code, making its integration in the rest of the application very difficult due to lack of abstraction.

In this paper, we promote the use of an executable formal approach called Interactive Cooperative Objects (ICOs) within the high-fidelity prototyping phase of interactive systems development. This formal approach solves some of the limitations of Rapid Application Development (RAD) techniques currently used for high-fidelity prototyping. Indeed, it provides abstraction through models, rapid execution through simulation and testing through generation of test cases and scenarios. In addition, when the prototyping phase is terminated, the outcome is not only a partially running prototype, but also a partial formal description of its behaviour that can then be passed on to the development team in charge of the development of the final system to be deployed. Previous work we have done in that domain was focusing on the rapid prototyping of the interactive application [17]; our current work addresses the 3 levels of interactive systems prototyping: interaction technique level (including multimodal interactions with non standard input devices as tactile screens), interactive component (including sophisticated widgets such as range sliders of semi-transparent pop-up menus) [16] and the interactive application in complex environment as cockpits (both military and civil [1]), grounds segment for satellite control rooms [20] and Air Traffic Management interactive applications.

This paper focuses on the use of the ICOs formal description technique to support rapid prototyping of interaction techniques. More precisely, it presents how an interaction technique can be defined and then how it can "rapidly" evolve according to users' feedback and users' performance. Indeed, the tool support environment for ICOs (called PetShop) has been now extended to provide additional facilities such as model-based logging of events and state-changes to support usability evaluation activities classically imbricated with rapid prototyping. This paper also addresses how logging support can be used to carry out performance analysis of the interaction technique thus limiting user testing to interaction techniques that have been previously formally analysed.

This paper is organized as follows. Next section presents some related work and research questions in the field of model-based approaches for interactive systems. The ICO notation is described in section 3. Section 4 presents the CASE tool Petshop which allows editing and execution of ICO models. Section 5 presents, on two small examples, how prototyping can be managed with PetShop and ICOs. Section 6 concludes the paper.

2 Model-Based Approaches for Interactive Systems

When formal methods were initially used for interactive systems [16], models were limited to the dialog part, making them less prominent for runtime use as only one part of the interactive system was taken into account. In order to address issues raised by real life application, current trend in interactive systems engineering is to develop models for all the parts of the systems.

Another parallel track of research work has been targeting at modelling new interaction techniques in order to be able to deal with current practice in the field of HCI. To deal with WIMP and post-WIMP interaction techniques, several notations have been proposed from Data-flow-based notations such as Wizz'ed [7], Icon [6], Nimmit [23] or InTml [8] to event-based notations such as Marigold [23], Hynets [23] or ICO [8]. Hybrid models integrating both event-based and data-flow-based notations have also been presented in [8] and in [15]. With respect to that later work, the work presented here extends the work presented in [15] by removing the data-flow model dealing with input devices configuration and proposing a single event-based notation described in the next section.

The work presented in this paper is about providing a modelling technique capable of representing the behaviour of an entire interactive application (from physical to functional interaction) using a dedicated Petri net dialect. It also targets at new interaction techniques (e.g. multimodal, direct manipulation ...) such as the ones used in the field of HCI.

This paper shows how the CASE tool Petshop [1] embeds the system models (which represent an interactive system from the interaction technique through to the system functional core) using the ICO notation at runtime for:

- Prototyping of models
- Execution of application to check
- Analysis as a way of supporting models construction by providing additional information about the properties of the models under construction.

3 The ICO Formalism

The ICO formalism is a formal description technique dedicated to the specification of interactive systems [19]. It uses concepts borrowed from the object-oriented approach (dynamic instantiation, classification, encapsulation, inheritance, client/server relationship) to describe the structural or static aspects of systems, and uses high-level Petri nets [23] to describe their dynamic behavioral aspects.

3.1 Cooperative Object

The ICO notation depends on Cooperative objects, A Cooperative Object states how the object reacts to external *stimuli* according to its inner state. The COs behaviour is called the Object Control Structure (ObCS) is expressed in a language based on Object Petri Net (OPN) (see Fig. 1.). An ObCS can have multiple places and transitions that are linked with arcs like standard Petri nets. As an extension to these standard arcs, ICO provides additional input arcs: Test arcs and Inhibitor arcs. Each place has an initial marking (represented by one or several tokens in the place) describing the initial state of the system.

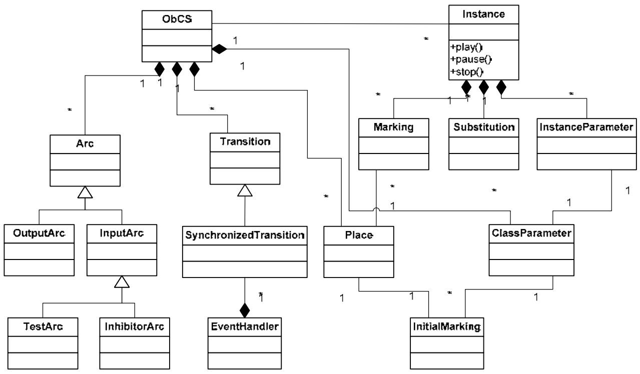


Fig. 1. Metamodel of the COs exhibiting runtime features

With respect to "standard" Petri nets, the object-oriented nature of the Cooperative Objects supports instantiation. Indeed, every ObCS can be instantiated and allows multiple executions of the same class as in object oriented programming languages. These instances can be parameterised by constructor arguments. This parameterisation is used to associate markings to the Petri net describing the behaviour of the instantiated Cooperative Object. For example, in a case of a multiple mouse interaction (i.e. in interactive cockpits such as the Airbus A380), each mouse driver is a distinct instance of an ObCS class with different Class Parameters (i.e. the number of the mouse) and so the behaviour model of each driver handle its own coordinates represented in the marking of the instance. For more details about that type of modelling see [1]. Fig. 1 presents a subset of the class diagram of ICOs. As stated above, the main element used for prototyping is related to the fact that each class can have several instances (as shown on the right-hand side of the figure) and that instances can be Played, Paused or Stopped.

3.2 Interactive Cooperative Objects

To allow dealing with the specificities of interactive systems the Cooperative Objects formalism has been extended. The resulting notation is called Interactive Cooperative Objects.

An ICO is a 6-tuple <CO, S_u, *Wid*, *Event*, *Act*, *Rend*> where:

- CO is a Cooperative Object described in section 3.1,
- S_u is a set of user services (a user service is a set of synchronized transitions),
- *Wid* is a set of interactive widgets (e.g. buttons, listbox, …) linked to the ICO class,
- *Event* is a set of user events coming from items of *Wid,*
- *Act* and *Rend* are the activation and rendering function described below.

Act: An activation function defines the relationship between events triggered by users while interacting with user interface objects (by manipulation of input devices such as mouse, keyboard, voice recognition systems …) with the transitions of the ObCS. When an event is triggered the related transition can be fired if the transition was fireable (according to the current marking of the Petri net).

Rend: A rendering function defines how the state changes in the ObCS influence the changes in the presentation (what the user perceives of the application). The state changes are linked to the entering in or exiting of a token in a place.

4 Prototyping of ICO Models Using Petshop Tool

To support the manipulation of the ICO notation, a CASE tool called Petshop [1] has been developed. It includes a Java implementation of a Object-oriented Petri net interpreter and some analysis tools for verifying properties on the models. The tool is publicly available at http://ihcs.irit.fr/petshop.

4.1 Structure

Fig. 2 represents the high level structure of Petshop. In Petshop, it is possible to edit, execute and analyze the instances of ObCS. When the user edits an instance, Petshop starts to update the ObCS (the class) and then updates all the instances of this class. During the first execution of the instance, the instantiation engine takes the ObCS to create an instance. Next, this instance is executed and can be directly managed by the user of Petshop (started, paused and stopped). When the instance is running, Petshop can also analyze the model (currently limited to the calculation of place invariants and transition invariants [10]). An example of PetShop user interface is presented in Fig. 2.

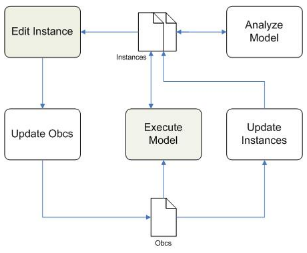


Fig. 2. High Level Structure of Petshop

4.2 Edition of Models

The CASE tool Petshop allows:

- to graphically add Petri net items (place, transition and different arcs) ,
- to modify the initial construction parameters of the class (e.g. editing a set of variables that may have different values for each instantiation)
- to modify the initial marking for each place (that corresponds to raw values or to references to the initial parameters of the class),
- to change the executable code in the transition,
- to modify the layout of the Petri net,
- to cut copy paste part of the model,
- to undo redo any change,
- to navigate through large models via mini map or through a large set of models via a tree.

4.3 Execution of Models

In Petshop a toolbar ( ) allows the user to start/stop/pause an instance of the ObCS. There are two modes of execution of instances:

- A normal execution in which the user is a spectator of execution and observes the execution of an instance. Transitions are fired using random enabling substitutions,
- A step by step execution in which the user can select a substitution to fire the transition.

  At runtime, the execution of instances gives the following feedback to the user:
- The marking is shown by the number of tokens present in a place,
- The fireability of transitions is shown by colour changes: purple for fireable or gray for not fireable,
- The firing of a transition and the updating of the marking (by the evolution of tokens in the input and output places of the fired transition).

Petshop also provides observability and controllability services via an API for externals programs (in our case the window manager of the plateform handing input devices). Observability services send events to subscribers when: markings change, substitutions change and events are raised in code associated to the transitions. Controllability services receive events from external sources and fire the related transition of a user service. All traces of execution can be logged to an external file allowing further analysis such as usability evaluation of the interactive systems [5].

5 Prototyping Interactive Systems with ICOs

This section presents the prototyping capabilities of PetShop and the ICO notation. These capabilities are presented on two examples extracted from case studies. They show different aspects illustrating how prototyping can be performed at different levels of the architecture of interactive systems.

5.1 Prototyping Interaction Techniques

The example in this section presents how it is possible using the ICO notation to prototype low level interaction techniques. Such prototyping is critical to increase usability of interactive applications as fine tuning of interaction can have a huge impact on the overall performance of users [13].

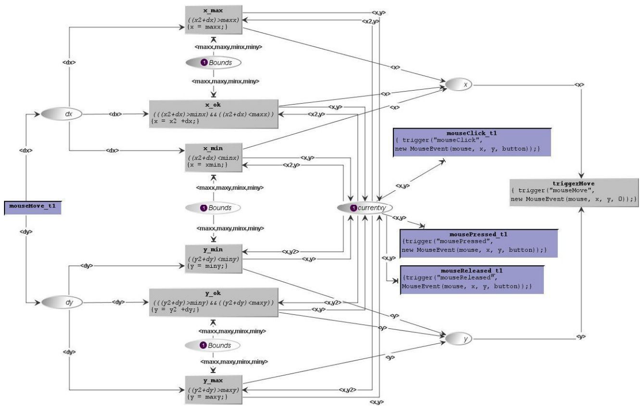


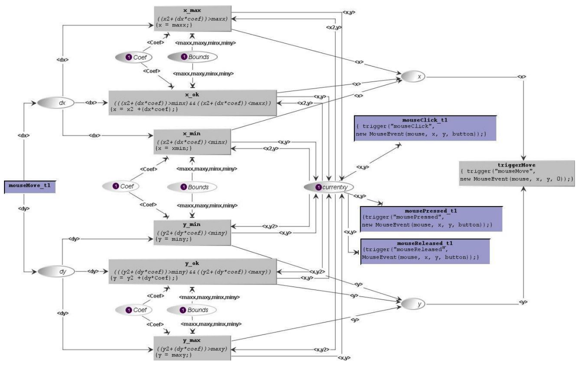
Fig. 3. ICO model of a mouse driver

The model of Fig. 3 describes a transducer for handling low level events. It models how events from the input device (in such as a case a pointing device like a mouse) are received from the input device and how they are transformed according to the need of the interactive application.

Dark transitions represent the transitions that are available according to the current marking of the model. Their black border means that they are connected to events i.e. even though they are available according to the current marking, they must additionally receive an event to be actually fired. The model can receive 4 different events: *mouseMove*, *mousePressed*, *mouseReleased* and *mouseClick*. The current position of the cursor of the input device is stored in the place *Currentxy*. When a *mouseMove* event is received the transducer has to transform the *dx, dy* parameters received in *x* and *y* position to reflect that change on the mouse cursor. In order to keep the cursor inside a set of predefined bounds (this could be for instance the size of the screen or the size of a portion of a window) the transformation of x and y values according to dx and dy parameters has to be constrained. This is the role of the places named *Bounds*. As for a notational aspect these places are virtual places i.e. virtual copies of a single place. This notational aspect is used to reduce the number of arcs when the same place is connected to many transitions.

The code of the transitions *mouseClick*, *mouseReleased* and *mousePressed* feature contain the *Trigger* construct. This means that, when one of these transitions is fired the model will raise an event. Other models registered to the current model will then be notified for each event triggered.

The model in Fig. 4 shows how the previous model can be modified according to requests from modification (after usability evaluation for instance).



```
x_max
((x2+(dx*coef))>maxx)
{x = maxx;}

x_ok
(((x2+(dx*coef))>minx)&&((x2+(dx*coef))<maxx))
{x = x2 +(dx*coef);}

x_min
((x2+(dx*coef))<minx)
{x = xmin;}

y_min
((y2+(dy*coef))<miny)
{y = miny;}

y_ok
(((y2+(dy*coef))>miny)&&((y2+(dy*coef))<maxy))
{y = y2 +(dy*Coef);}

y_max
((y2+(dy*coef))>maxy)
{y = maxy;}

mouseClick_t1
{ trigger("mouseClick",
new MouseEvent(mouse, x, y, button));}

triggerMove
{ trigger("mouseMove",
new MouseEvent(mouse, x, y, 0));}

mousePressed_t1
{trigger("mousePressed",
new MouseEvent(mouse, x, y, button));}

mouseReleased_t1
{trigger("mouseReleased",
MouseEvent(mouse, x, y, button));}
```

Fig. 4. Modified ICO model of a mouse driver (acceleration of mouse move events)

The modification includes a new element in the interaction technique: the acceleration. Indeed, the movements on the table where the mouse is located are typically much more constrained than the virtual space available to the cursor. For this reason mouse drivers will embed an acceleration mechanism that increase cursor movement according to speed. This is modelled by adding the places *Coef* in the models and connecting them to the transitions in charge of the calculation of the new position of the cursor. The code of these transitions shows that *dx* and *dy* parameters are multiplied by the coefficient (stored in the token of the place *Coef*).

5.2 Prototyping Applications

While the prototyping of interaction techniques is critical for fine tuning of interaction, prototyping is also needed at a higher level. This section presents how PetShop and ICO support prototyping at the dialogue level of interactive applications.
The prototyping aspects remain the same as for the interaction technique i.e. models describing the behaviour of the applications at the dialogue level can be interactively modified and the impact of the modifications can be immediately perceived.

The application under consideration here is called MPIA. The Multi Purpose Interactive Application (MPIA) is an application available in the cockpits of several aircrafts that aims at handling several flight parameters. It is made up of 3 pages (called WXR, GCAS and AIRCOND). The WXR page is responsible for managing weather radar information; GCAS is responsible for the Ground Anti Collision System parameters while AIRCOND deals with settings of the air conditioning. Due to space constraints we don't present in details the interactive modifications of the models but the interested reader can see detailed behaviour of that application (in a reconfiguration process after hardware failure in a cockpit) in [18].

6 Conclusion

This paper presents the ICO notation for the description of interactive systems via graphical models which can be edited and executed at runtime. The ICO notation, an extension of object Petri nets has a dedicated CASE tool called Petshop. This runtime capability increases the possibilities of modelling by supporting prototyping, testing, and verification. This paper presented how prototyping of interactive applications can be performed at two different levels: interaction technique and dialogue model. The later is extracted from an industrial example dealing with cockpit applications in civil aircrafts. We have studied the usability of ICOs and PetShop for prototyping phases in an informal with software engineers involved in the field of Air Traffic Control applications [2]. Informally we can report that modification of models was fine while creation of models and connecting models was not performed in a satisfying way. Testing of the tool is available at http://ihcs.irit.fr/petshop. The specific application area that we consider in the paper is ground segment applications for satellite control, but the results have been applied and are applicable to other application areas with similar requirements.

Acknowledgements. This work is supported by the EU funded Network of Excellence ResIST http://www.resist-noe.eu contract n°026764 and the CNES funded R&T Tortuga project http://ihcs.irit.fr/tortuga/ contract n° R-S08/BS-0003-029. We would also like to thanks the reviewers for their in-depth thoughtful comments.

References

1. Barboni, E., Navarre, D., Palanque, P., Basnyat, S.: Addressing Issues Raised by the Exploitation of Formal Specification Techniques for Interactive Cockpit Applications. In: HCI Aero 2006, p. t.b.p., Seattle (2006)
2. Bastide, R., Navarre, D., Palanque, P.: A Tool-Supported Design Framework for Safety Critical Interactive Systems. Interacting with computers 15(3), 309–328 (2003)
3. Bastide, R., Palanque, P., Duc, L.: Integrating Rendering Specifications into a Formalism for the Design of Interactive Systems. In: DSV-IS 1998, pp. 171–190 (1998)
4. Beck, K.: Extreme Programming Explained: Embrace Change. Addison-Wesley, US (1999)
5. Bernhaupt, R., Navarre, D., Palanque, P., Winckler, M.: Model-Based Evaluation: A New Way to Support Usability Evaluation of Multimodal Interactive Applications In Maturing Usability, Quality in Software, Interaction and Value. In: Human-Computer Interaction Series, pp. 96–119. Springer, Heidelberg (2007)
6. Dragicevic, P., Fekete, J.-D.: Input Device Selection and Interaction Configuration with ICON. In: Proceedings of IHM-HCI 2001, People and Computers XV - Interaction without Frontiers, pp. 543–448. Springer, Heidelberg (2001)
7. Esteban, O., Chatty, S., Palanque, P.: Whizz'Ed: a visual environment for building highly interactive interfaces. In: Proceedings of the Interact 1995 conference, pp. 121–126 (1995)
8. Figueroa, P., Green, M., Hoover, J.: InTml: A Description Language for VR Applications. In: Proceedings of Web3D 2002, Arizona, USA, pp. 53–58 (2002)
9. Fowler, M., Highsmith, J.: The Agile Manifesto. Software Development (August 2001)
10. Genrich, H.J.: Predicate/Transitions Nets. In: Jensen, K., Rozenberg, G. (eds.) High-Levels Petri Nets: Theory and Application, pp. 3–43. Springer, Berlin (1991)

11. Gulliksen, J., Goransson, B., Boivie, I., Blomkvist, S., Persson, J., Cajander, A.: Key principles for user-centred systems design. Behaviour and Inf. Tech. 22, 397–409 (2003)
12. Jacob, R.: A Software Model and Specification Language for Non-WIMP User Interfaces. ACM Transactions on Computer-Human Interaction 6(1), 1–46 (1999)
13. Kabbash, P., Buxton, W.A.: The "prince" technique: Fitts' law and selection using area cursors. In: Proceedings of the ACM CHI Conference, pp. 273–279. ACM Press, New York (1995)
14. Lim, Y., Pangam, A., Periyasami, S., Aneja, S.: Comparative analysis of high- and low-fidelity prototypes for more valid usability evaluations of mobile devices. In: Proc. of NordiCHI 2006, vol. 189, pp. 291–300. ACM, New York (2006)
15. Navarre, D., Palanque, P., Dragicevic, P., Bastide, R.: An Approach Integrating two Complementary Model-based Environments for the Construction of Multimodal Interactive Applications. Interacting with Computers 18(5), 910–941 (2006)
16. Navarre, D., Palanque, P., Bastide, R., Sy, O.: Structuring interactive systems specifications for executability and prototypability. In: Palanque, P., Paternó, F. (eds.) DSV-IS 2000. LNCS, vol. 1946, pp. 97–120. Springer, Heidelberg (2001)
17. Navarre, D., Palanque, P., Bastide, R., Sy, O.: A Model-Based Tool for Interactive Prototyping of Highly Interactive Applications. In: 12th IEEE International Workshop on Rapid System Prototyping, Monterey, USA, IEEE, Los Alamitos (2001)
18. Navarre, D., Palanque, P., Basnyat, S.: Usability Service Continuation through Reconfiguration of Input and Output Devices in Safety Critical Interactive Systems. In: Harrison, M.D., Sujan, M.-A. (eds.) SAFECOMP 2008. LNCS, vol. 5219, pp. 373–386. Springer, Heidelberg (2008)
19. Navarre, D., Palanque, P., Bastide, R., Schyn, A., Winckler, M., Nedel, L.P., Freitas, C.M.D.S.: A model-based approach for engineering multimodal interactive systems. In: Costabile, M.F., Paternó, F. (eds.) INTERACT 2005. LNCS, vol. 3585, pp. 170–183. Springer, Heidelberg (2005)
20. Palanque, P., Bernhaupt, R., Navarre, D., Ould, M., Winckler, M.: Supporting Usability Evaluation of Multimodal Man-Machine Interfaces for Space Ground Segment Applications Using Petri net Based Formal Specification. In: Ninth International Conference on Space Operations, CD-ROM proceedings, Rome, Italy, June 18-22 (2006)
21. Parnas, D.L.: On the use of transition diagram in the design of a user interface for interactive computer system. In: Proceedings of the 24th ACM Conference, pp. 379–385 (1969)
22. Peterson, J.L.: Petri Net Theory and the Modeling of Systems. Prentice-Hall, Englewood Cliffs (1981)
23. Reason, J.: Human Error, 302 pages. Cambridge University Press, Cambridge (1990)
24. Rettig, M.: Prototyping for tiny fingers. Commun. ACM 37(4), 21–27 (1994)
25. Risoldi, M., Amaral, V.: Towards a Formal, Model-Based Framework for Control Systems Interaction Prototyping. Rapid Integration of Software Engineering Techniques, 144–159 (2007)
26. Rudd, J., Stern, K., Isensee, S.: Low vs. high-fidelity prototyping debate. Interactions 3(1), 76–85 (1996)
27. Vanacken, D., De Boeck, J., Raymaekers, C., Coninx, K.: NiMMiT: a Notation for Modelling Multimodal Interaction Techniques. In: International Conference on Computer Graphics Theory and Applications, Portugal (2006)
28. Wieting, R.: Hybrid High-Level Nets. In: Proc. of the 1996 Winter Simulation Conference, pp. 848–855. ACM Press, New York (1996)
29. Willans, J.S., Harrison, M.D.: Prototyping pre-implementation designs of virtual environment behaviour. In: Nigay, L., Little, M.R. (eds.) EHCI 2001. LNCS, vol. 2254, pp. 91–108. Springer, Heidelberg (2001)