A generic framework for executable gestural interaction models

Romuald Deshayes, Tom Mens, Philippe Palanque

A Generic Framework for Executable Gestural Interaction Models

Romuald Deshayes, Tom Mens
Service de Génie Logiciel, Université de Mons
7000 Mons, Belgique
Email: romuald.deshayes,tom.mens@umons.ac.be

Philippe Palanque
ICS-IRIT, Université Paul Sabatier
31062 Toulouse, France
Email: palanque@irit.fr

***Abstract*—Integrating new input devices and their associated interaction techniques into interactive applications has always been challenging and time-consuming, due to the learning curve and technical complexity involved. Modeling devices, interactions and applications helps reducing the accidental complexity. Visual modeling languages can hide an important part of the technical aspects involved in the development process, thus allowing a faster and less error-prone development process. However, even with the help of modeling, a gap remains to be bridged in order to go from models to the actual implementation of the interactive application. In this paper we use ICO, a visual formalism based on high-level Petri nets, to develop a generic layered framework for specifying executable models of interaction using gestural input devices. By way of the CASE tool Petshop we demonstrate the framework's feasibility to handle the Kinect and gesture-based interaction techniques. We validate the approach through two case studies that illustrate how to use executable, reusable and extensible ICO models to develop gesture-based applications.**

I. Introduction

Gestural interaction was proposed several decades ago [1] but it took until now for unconstrained gestural interaction to become widely available in the mass market, with the release of new input devices such as Kinect, Wiimote and multi-touch sensing surfaces. With the release of Microsoft's hands-free controller Kinect, a consumer 3D sensor packed with powerful real-time algorithms to track a user's body, numerous novel interaction applications were published on the internet [2], [3].

Developing reactive interactive applications involves a lot of technical complexity, because developers need to address all low-level aspects regarding input and output devices. Beyond that, the toolkits and programming environments proposed by manufacturers integrate innovations only after they have raised enough interest to developers. We propose a generic reusable framework for gestural interaction that copes with this accidental complexity by raising the level of abstraction, and separating concerns into different modules. The framework is built using ICO, a powerful model-based approach relying on high-level Petri nets [4], and realized in Petshop, a tool for the specification and execution of interactive systems based on ICO models.

Our framework enables the specification of executable models that receive gestural input from devices such as the Kinect. The raw input data is converted into abstract events that are processed by the models and interpreted by virtual objects on an output device such as a graphical user interface or 3D rendering engine. This contribution counters the critique of [5] that model-based approaches "model the previous generation of user interfaces".

II. Case Studies

To illustrate the expressiveness and reusability of our framework, we present two different case studies. We refer to [6] for more technical details.

Our first case study is a Pong game using the Java Swing UI. The player needs to prevent the opponent (the computer or another human player) from returning the ball with his paddle. The ball can bounce on the upper and lower parts of the screen. A gestural interface allows the player to use hand gestures instead of a joystick. Kinect's 3D sensor is used to track a person's hand movements in real-time. The paddle on the screen follows the player's hand movements. The ball can be grabbed by closing the hand when the ball is close to the paddle. When the hand is reopened, the ball is released and the game continues.

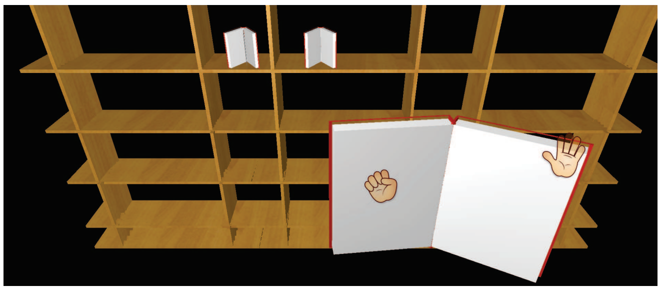
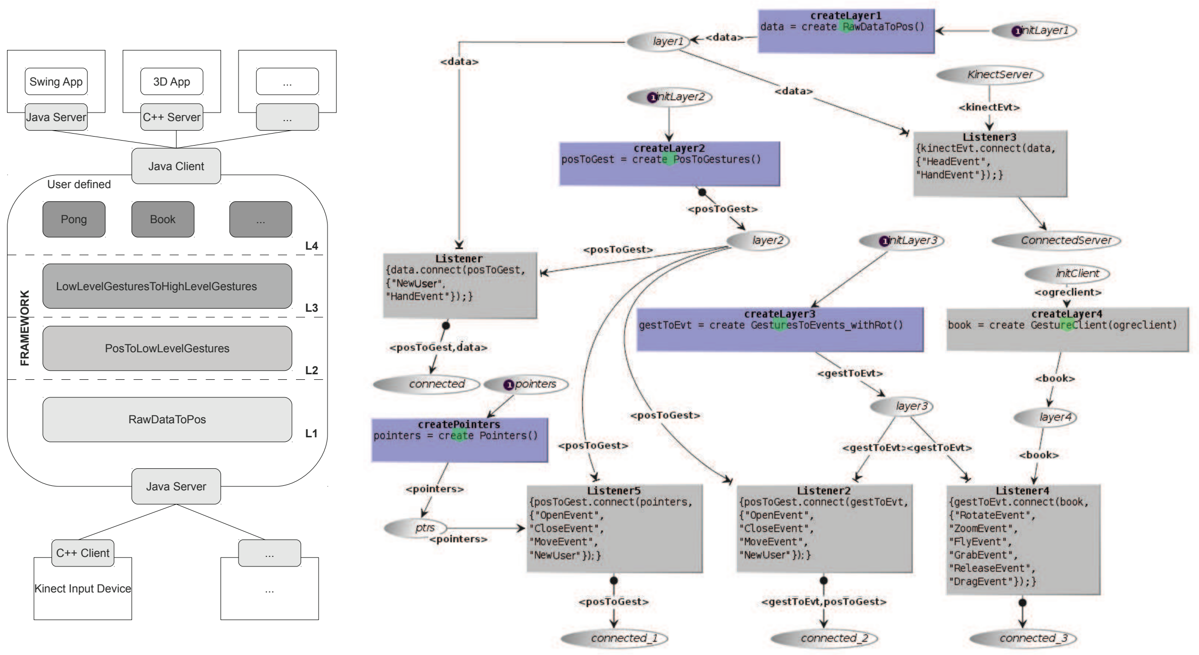


Fig. 1. Screenshot of the bookshelf application.

In the second case study, depicted in Fig. 1, the user gesturally interacts with a virtual bookshelf on the screen. By moving an open hand, the user can browse books stored on the shelf. He can also drag a selected book towards him. To open the book he has to move both closed hands away from each other. A drag from right to left (or left to right) will turn a page from the book. Approaching both closed hands will close the book, and a drag towards the screen will place the book back on the bookshelf. A YouTube video of this case study is available at http://youtu.be/m9NIvZpQyjs.

The Kinect comes along with the NITE framework, providing a set of algorithms to perform real-time body tracking. These algorithms are used to retrieve the position of the hands and head of the user in 3D space, the origin of the coordinate



```
createLayer1
data = create RawDataToPos()

createLayer2
posToGest = create PosToGestures()

Listener3
{kinectEvt.connect(data,
{"HeadEvent",
"HandEvent"});}

Listener
{data.connect(posToGest,
{"NewUser",
"HandEvent"});}

createLayer3
gestToEvt = create GesturesToEvents_withRot()

createLayer4
book = create GestureClient(ogreclient)

createPointers
pointers = create Pointers()

Listener5
{posToGest.connect(pointers,
{"OpenEvent",
"CloseEvent",
"MoveEvent",
"NewUser"});}

Listener2
{posToGest.connect(gestToEvt,
{"OpenEvent",
"CloseEvent",
"MoveEvent",
"NewUser"});}

Listener4
{gestToEvt.connect(book,
{"RotateEvent",
"ZoomEvent",
"FlyEvent",
"GrabEvent",
"ReleaseEvent",
"DragEvent"});}
```

Fig. 2. Layered architecture of the gestural interaction framework (left). Communication between layers following the Observer design pattern (right).

system being located on the eye of the Kinect's camera. As the provided algorithms do not allow knowing whether hands are open or closed, we implemented this feature to widen the range of possible gestures to be recognized.

III. Layered Architecture

Our gestural interaction framework is implemented following a layered client-server architecture (Fig. 2 left). All framework layers are specified as executable ICO models.[1] Gestural input devices implement a simple client connected to a Java server that forms the entry point to the lowest layer of the framework. The Java server sends the raw data to this layer, which processes the information and sends it further to layer 2 and so on. Information that has been processed in layer 4 is sent by another Java server outside the framework to the target client application. Target applications can be developed using different output devices. For the first case study, we have used a Java Swing user interface, and for the second we have used C++ client code interfacing to the Ogre3D rendering engine.

Communication between layers is explicitly modeled by an Observer design pattern (Fig. 2 right). This model is executed only once at startup and is used to create the layers and establish the communication between them. It allows each layer to receive and process events sent by the lower layers. Transitions containing the `create` primitives (depicted by a small green arrow inside the transition) consume a token to create a new model instance. For example, transition `createLayer1` produces a token containing a reference to the model of layer 1 and puts it in place `layer1`. The models of communicating layers are connected by a listener that specifies which events generated by the sending layer will be observed by the receiving layer. For example, transition `Listener` specifies that layer 2 listens to the events `NewUser` and `HandEvent` of layer 1.

Layer 1 is used to detect new users and propagate them through the rest of the framework, but also to receive raw information from the gestural input device about the hands and head of the current users and combine them to calculate the hands' positions relative to the head. Layer 2 transforms absolute positions of the hands into relative movements between two consecutive updates. It is in this layer that state changes of the hands (opened or closed) are detected.

Layer 3 combines low-level gestures of layer 2 (e.g., Open, Close, Move) into high-level ones (e.g., Move, Drag, ColinearDrag and NonColinearDrag). This layer is divided into 2 models. The first model (Fig. 3) serves to process the state of the users' hands. The second model combines low-level gestures into high-level ones by taking into account the state of the user's hands (e.g., open or closed). The possible states are represented in Fig. 3 by 4 different places: `twoOpened`, `twoClosed`, `leftClosed`, `rightClosed`. Transition `NewUser_` processes the incoming event *NewUser* to consume a token from place `players` and associates an id to it when a new user joins the game. The system initializes each new user with both hands open, and will produce a token in place `twoOpened`. When an *open* (resp. *close*) event is received from layer 2, transition `OpenEvent_` (resp. `CloseEvent_`) is triggered, producing a new token in place `openTokens` (resp. `closeTokens`). Depending on the state of the user, one out of 8 possible transitions can be fired to change the user state. Imagine that a user enters the game with both hands opened. A token will be generated for his id in place `twoOpened`. If he closes the left hand, a *close* event is received and the transition `CloseEvent_` is fired, thus producing a token in place `closeTokens`. The only

[1]For more details about this, consult to technical report at http://difusion.academiewb.be/vufind/Record/UMONS-DI:oai:di.umons.ac.be:17510.

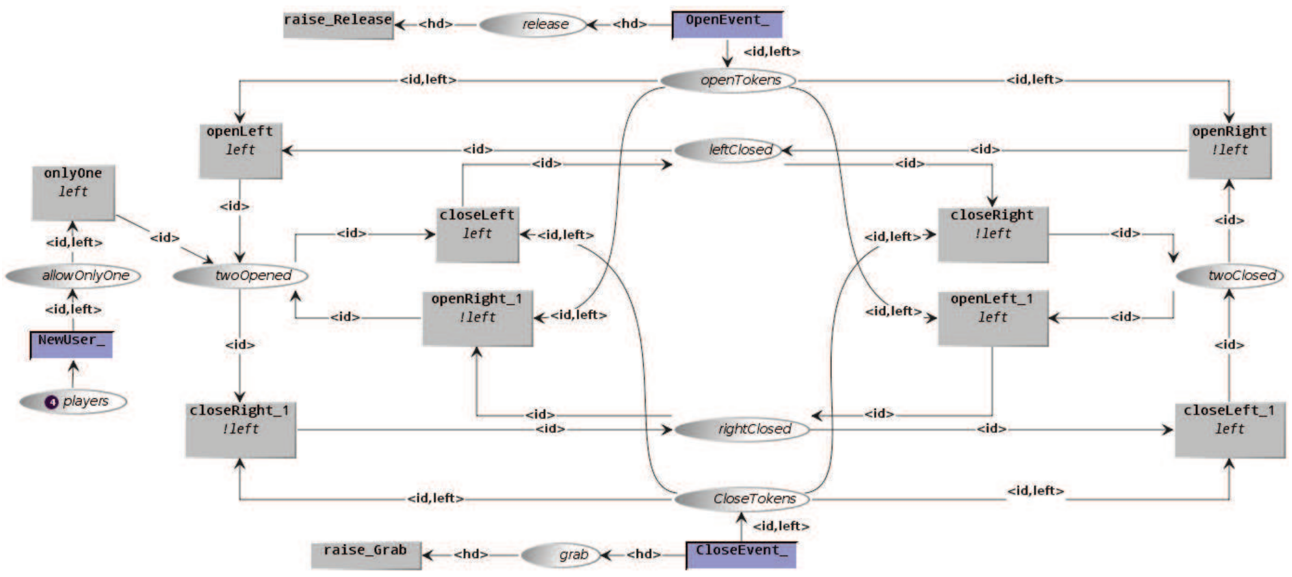


Fig. 3. Layer 3: Model processing the state of the users' hands.

transition that can then be fired is `closeLeft`. This transition consumes both tokens and produces a new token for the user's id in place `leftClosed`.

Layer 4 contains a different model for each type of object in the virtual scene the user is interacting with. These objects can be controlled using the gestures interpreted by layer 3 (Fig. 4). The same gesture can be interpreted differently by different objects, e.g., picking up an object lying on a desk or opening a door. By combining the provided gestures, one can create a wide range of interaction behaviors. This versatility is illustrated through the case studies of Section 2.

IV. Lessons Learned

We can learn a number of lessons from using our visual formalism based on Petri nets for developing a gestural interaction framework and having applied it to two case studies. The visual formalism reduces the technical complexity of developing gestural interaction behavior by raising the level of abstraction. We have personally experienced that hardcoding such behavior using standard textual programming languages can be quite cumbersome. A visual formalism facilitates system understanding as its behavior is highly concurrent within models and as several models evolve in parallel. The dynamic simulation facilities allow finding and correcting bugs more easily and offer an elegant way of bridging the gap between modeling and execution. When executing the models in Petshop we can actually see the Petri net tokens moving from one place to another, and see their values changing as different transitions are fired. We can even change models during their execution, and study how this affects the running process. One very useful feature is the ability of models to communicate by means of events, using synchronized transitions in the listening models to trigger transitions only when an event is received. Thanks to the ability to store data in tokens, we were able to process hand state and position in a very straightforward way. This data could be used for calculations in the transitions, thus resulting in an increased expressiveness.

The work presented in this article has convinced us that executable visual modeling is the way to go for developing interactive applications. Not only is it adapted for this kind of systems, but also the learning time is relatively low. Without any prior knowledge of Petri nets or ICO models, it took the first author about 4 weeks to learn the formalism and the Petshop tool, and to design a first running version of our framework. After that, it turned out to be very easy to add more functionality (such as pointers and new gestures) to our models. Excluded from the aforementioned learning time is the time needed to understand how to use and interface with the gestural input and graphical output devices.

An alternative for executable specification of gestural interaction is the use of heterogeneous modeling. In [7] we explored this alternative by developing a similar layered client-server framework using the ModHel'X environment [8]. The different layers were specified using different visual formalisms such as synchronous data flow models and timed finite state machines. The challenge of heterogeneous modeling lies in the need for semantic adaptation between the different layers using different formalisms, as well as the need to master multiple formalisms.

A limitation of our approach is the potential difficulty for domain experts and designers of interactive applications to master the notation of ICO models. It is important to note, however, that the low-level models are device-dependent and only have to be built once. Designers then only need to adapt the models to implement the desired behavior for the target interaction technique. Beyond that, we believe that the full expressive power of ICO is not always needed and can be abstracted away by using a domain-specific modeling language. At the risk of a certain loss of expressiveness, this would allow designers to compose gestures and process them to create new behaviors for virtual objects more easily, without requiring detailed knowledge of the underlying visual formalism.

A current limitation is the performance issues encountered during our case studies, because Petshop interprets and simulates ICO models at runtime. To be able to apply it in real commercial applications, faster execution is necessary. This can be achieved by compiling the models directly into executable code, but it will go at the expense of no longer being able to dynamically visualize and modify the models during their execution, which is very useful for development and debugging purposes.

In [9], ICO models were proposed to model multimodal interactions in virtual reality applications. As a proof-of-concept, a virtual chess game was developed that could be manipulated by a single user using a data glove on one hand. The design of this application was quite different from ours. It did not focus on reusable models, did not include the state of the user or the notion of virtual objects, and contained a very limited set of gestures using only one hand.

According to [5], one of the limitations of many virtual environment toolkits is the predefined and limited small amount of interaction means they provide, which are intended to be used regardless of context. To extend the flexibility of such toolkits, developers must be provided with the possibility to design, test and verify new interaction techniques. The authors presented Marigold, a toolset supporting such a development process. This toolset allows visual specification of the interaction techniques but unlike our approach, it is not dynamic. C code is generated from the models, making it impossible to modify the models at runtime or to see the models running. Moreover, the entire specification resides in a single monolithic model, unlike our layered architecture that is more modular and easier to adapt.

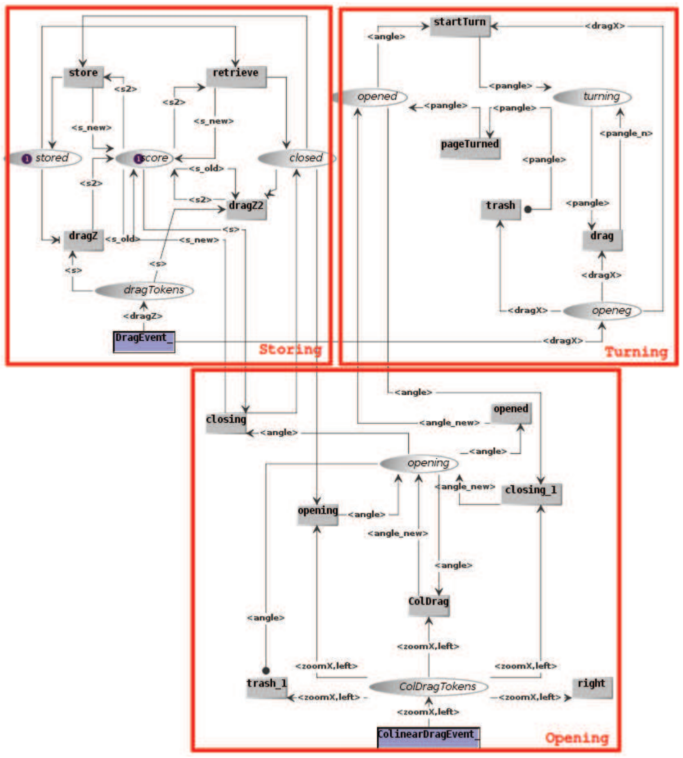


Fig. 4. Layer 4: ICO model of the interaction behavior for a virtual book.

V. Conclusions and Future Work

We implemented a generic, layered and modular client-server framework (see YouTube video http://youtu.be/m9NIvZpQyjs) to specify and execute models of gestural interaction based on the visual formalism of high-level Petri nets. This framework relieves the developer from writing complex and statically compiled code. Our case studies illustrated the feasibility of using the framework to allow a user to interact with multiple virtual objects displayed on screen.

The models in each framework layer are based on discrete events provided by the gestural input devices. High-level Petri nets, incarnated as ICO models in Petshop, proved to be particularly suited for expressing such models. One of the reasons was their ability to concurrently execute complex behavior involving a dynamically changing number of actors (e.g., players, virtual objects) and requiring a huge amount of user interaction. Using high-level Petri nets allowed us to cope with the high complexity by separating the behavior of gestural interactions in separate communicating layers. In addition, data manipulation and data flow were facilitated by the ability to encapsulate data in tokens. Future work will be carried along the lines discussed in the previous section. We plan to validate and extend the framework further with more extensive case studies and other input devices such as the Wiimote. We plan to define a domain-specific language on top of ICO as a means to provide the scaffolding required for wider use by designers and developers.

Acknowledgment

The first author is financed by a FRIA scholarship.

References

[1] R. A. Bolt, "Put-that-there: Voice and gesture at the graphics interface," in *Proc. SIGGRAPH*. ACM, 1980, pp. 262–270.

[2] G. Bailly, R. Walter, J. Müller, T. Ning, and E. Lecolinet, "Comparing free hand menu techniques for distant displays using linear, marking and finger-count menus," in *INTERACT*, 2011, pp. 248–262.

[3] Z. Ren, J. Meng, J. Yuan, and Z. Zhang, "Robust hand gesture recognition with Kinect sensor," in *ACM Int'l Conf. Multimedia*, 2011, pp. 759–760.

[4] D. Navarre, P. Palanque, J.-F. Ladry, and E. Barboni, "ICOs: A model-based user interface description technique dedicated to interactive systems addressing usability, reliability and scalability," *ACM Trans. Comput.-Hum. Interact.*, vol. 16, no. 4, pp. 1–56, Nov. 2009.

[5] P. N. Sukaviriya, S. Kovacevic, J. D. Foley, B. A. Myers, D. R. Olsen Jr., and M. Schneider-Hufschmidt, "Model- based user interfaces: What are they and why should we care?" in *Proc. UIST94*. ACM, Nov. 1994, pp. 133–135.

[6] R. Deshayes, T. Mens, and P. Palanque, "Petrinect: A tool for executable modeling of gestural interaction," in *Proc. VL/HCC*, 2013.

[7] R. Deshayes, C. Jacquet, C. Hardebolle, F. Boulanger, and T. Mens, "Heterogeneous modeling of gesture-based 3D applications," in *MoDELS Workshops*, 2012.

[8] C. Hardebolle and F. Boulanger, "Modhel'x: A component-oriented approach to multi-formalism modeling," in *MoDELS Workshops*, 2007, pp. 247–258.

[9] D. Navarre, P. A. Palanque, R. Bastide, A. Schyn, M. Winckler, L. P. Nedel, and C. M. D. S. Freitas, "A formal description of multimodal interaction techniques for immersive virtual reality applications," in *INTERACT*, 2005, pp. 170–183.