

Very-High-Fidelity Prototyping for both Presentation and Dialogue Parts of Multimodal Interactive Systems

David Navarre, Pierre Dragicevic, Philippe Palanque, Rémi Bastide & Amélie Schyn

LIHS-IRIT, Université Paul Sabatier, F-31062 Toulouse Cedex, France
{dragice, navarre, palanque, bastide, schyn}@irit.fr
<http://lihs.irit.fr/{navarre, dragice, palanque, bastide, schyn}>

Abstract. This paper presents a tool suite (made up of two previously unrelated approaches) for the engineering of multimodal Post-WIMP Interactive Systems. The first element of this integration is ICoM (a data-flow model dedicated to low-level input modelling) and its environment ICoN which allows for editing and simulating ICoM models. The other element is ICoO (a formal description technique mainly dedicated to dialogue modelling) and its environment PetShop which allows for editing, simulating and verifying ICoO models. This paper shows how these two approaches have been integrated and how they support multimodal interactive systems engineering. We show on a classical rubber banding case study how these tools can be used for prototyping interactive systems. We also present in details how the changes in the interaction techniques impact the models at various levels of the software architecture.

Keywords. Interactive Systems Engineering, Multimodal interaction, Prototyping, CASE tools, Formal methods, formal description techniques; Post-WIMP.

Introduction

According to the recurring desire of increasing the bandwidth between the interactive system and the users more sophisticated interaction techniques called Post-WIMP have been proposed. However, the current contribution from the research community to the construction of such interactive systems remains at the level of working prototypes showing the feasibility and making empirical evaluation possible.

Recent contributions in the field of model-based approaches have been explicitly addressing this issue of coping with new interaction techniques. The aim of the work presented in this paper is to describe an approach (that is able to go beyond prototyping post-WIMP interaction techniques) fully integrated within interactive systems development. To this end we have integrated work done on low-level input management [0] with work on formal description techniques of dialogue models [0, 0].

Several notations have already proposed for dealing with post WIMP interaction techniques and for different kinds of applications. Data-flow-based notations such as Wizz'Ed [0] or ICoN [0] have been proposed for dealing with low-level flow of events

produced directly by input devices. This notion of flow has also been addressed with other notations where classical event and status based behaviours have been enhanced with continuous modelling such continuous Petri nets as in Marigold [0] or Hynets [0]. Higher-level models of this kind of interaction techniques have also been addressed using state-based notations as with basic Petri nets in [0] or with high-level Petri nets [0]. Early work in the field of multimodal interaction techniques has also addressed the aspects of fusion of modalities and a comparison of these work can be found in [0].

The paper is structured as follows. Section 2 presents the Input Configuration approach that is dedicated to low-level input handling in post-WIMP interactive systems. Section 3 recalls the Interactive Cooperative Objects formalism and its environment PetShop. In these sections, the two model-based approaches are exemplified on the same simple case study of the rubber banding interaction technique. Section 4 details a generic framework for the integration of these two approaches. Section 5 introduces a line drawing application exploiting the rubber banding interaction technique previously presented. The aim of this small case study is to show that the model-based approaches that we propose can deal completely with non standard interface components and innovative interaction techniques. This section presents also how to modify that case study to allow for multimodal (two handed) interaction. For space reasons, only such multimodal interaction technique is presented here while several others (including voice and gesture) have been dealt with in a similar way and presented at the conference.

Input-Configurations Modelling and Prototyping

ICON (Input Configurator) is a tool for designing input-adaptable interactive applications, i.e., applications that can be controlled with a wide variety of alternative input devices and techniques. ICON provides an interactive editor for the ICOM (Input Configuration Model) graphical notation. In this section, we give a brief overview of the ICOM notation and the ICON visual prototyping environment. More details on the notation and its associated tools can be found in [0, 0, 0].

Overview of the ICOM notation

The ICOM (Input Configuration Model) notation describes low-level input handling using interconnected modules, with reactive data-flow semantics. In this section, we briefly describe the main features and concepts behind ICOM.

Input Configurations

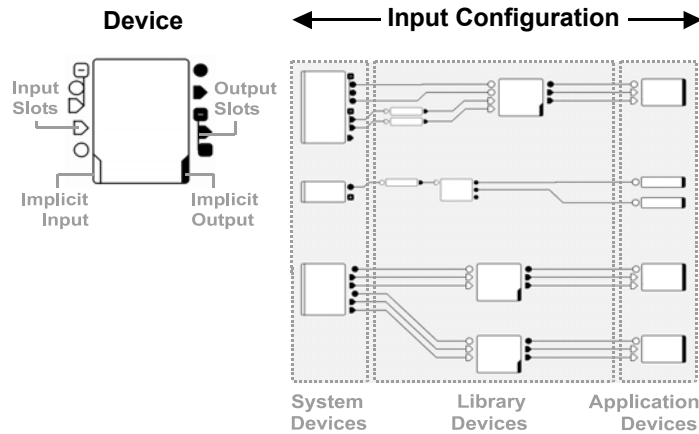


Fig. 1. Elements of the ICOM notation.

Devices and slots. ICOM’s main building blocks are *devices*, which are a broad generalization of input devices: ICOM devices can produce output values, but can also receive input values. Fig. 1 shows on the left the graphical representation of a device. A device has typed channels called *input slots* and *output slots*, each type having a distinct graphical representation (e.g., circle for Booleans, triangle for integers). Slots can be hierarchically grouped to form structured types, as shown on Fig. 1.

Implicit I/O. Whereas the basic behaviour of an ICOM device is processing input values into output values, alternative behaviour is shown on the device by the presence of “notches” (see Fig. 1). Non-deterministic devices are described as having *implicit input*, i.e., additional source of information not fully described by its set of input slots. Example of such devices include devices which are producing data on their own (physical input devices), or asynchronous devices which are temporally non-deterministic. Similarly, devices having *implicit output* produce alternative effects in addition to simply putting values on the output slots. Examples are devices that manipulate application objects, or devices producing graphical or sound feedback.

Connections. An input slot of a device can be linked to one or several compatible output slots of other devices by *connections*, which are represented by wires. ICOM’s execution model forbids multiple connections on the same input slot, as well as connections that generate cyclic dependencies.

Types of devices. There are three main categories of devices: *System devices* describe system resources such as input peripherals; *Library devices* are system-independent utility devices such as processing devices and adapters; *Application devices* are devices that control a specific application.

Input configurations. An *input configuration* is defined by a set of system and application devices, as well as a set of library devices and connections which map the system devices to the application devices.

ICON is modular, and subparts of an input configuration can be encapsulated into compound devices. For example, an input device and a feedback device can be connected then grouped to form a compound device having both external input and external output.

ICoM's Execution Model

Whereas the contract of a device is to update its output slots every time it is asked to, ICoM's execution model describes which devices must be triggered and when, and how values are propagated to other devices. The propagation mechanism used, described in [0], is very simple and effective.

ICoM's execution model follows the semantics of reactive synchronous languages such as Esterel [0] or Lustre [0], in which information propagation is conceptually instantaneous. In *reactive* systems, the environment (e.g., the source of input signals) is the master of the interaction, as opposed to conversational systems in which clients wait to be served. As a result, the way we handle input is closer from device drivers, which are reactive, than from event-driven mechanisms, which are intrinsically conversational.

Describing Interaction Techniques as Input Configurations

From ICoM's point of view, interaction techniques are transformation flows with feedback. Fig. 2 gives an example of scrolling through a document, and shows the feedback loop through implicit I/O. The Mouse device receives implicit input from the user, the Cursor device produces immediate feedback towards this user, and the Scrollbar tells the application to update its document view.

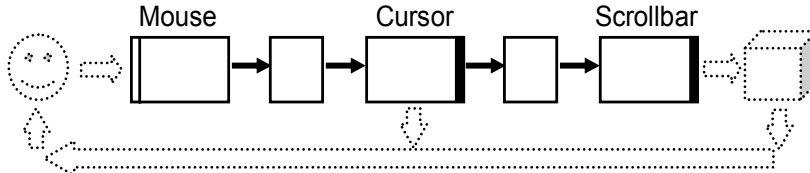


Fig. 2. Feedback flow while scrolling through a document

The ICON environment

The ICON (Input Configurator) Input Toolkit contains an extensible set of system devices and library devices for building input configurations. It provides a reactive machine for executing them, as well as a graphical editor for rapid prototyping. ICON is written in Java, and uses native libraries for managing input devices. In this section, we briefly describe the main features of ICON.

ICON Devices

System devices. ICON's system devices provide a low-level view of standard and alternative input devices. Under Microsoft Windows operating systems, ICON currently supports multiple mice, graphical tablets, gaming devices and 3D isometric

controllers, speech and gesture recognition, and MIDI controllers. System output devices are also available, such as Midi devices for playing music on soundcards, or speech synthesis devices.

Library devices. The ICON toolkit has a set of built-in utility devices including mathematical and boolean operators, signal processing devices, type and domain adapters, and devices for conditional control and dispatch. It also provides a set of graphical feedback devices such as cursors and semi-transparent components, which support overlay animation on top of Swing frames.

Toolkit devices. ICON provides a set of “Swing devices” for controlling existing Java applications that have no knowledge of ICON. One device allows generic control of any Swing widget by sending them mouse and keyboard events, whereas a set of widget-specific devices allow moving scrollbars programmatically or sending strings and caret commands to text components. Event dispatching strategies such as picking and focus are also encapsulated into individual devices.

Application devices. Developers can enhance controllability of their application by implementing devices that are specific to their application. Writing an application device is quite straightforward, and mainly requires declaring a set of input slots and implementing an “update” method which is automatically called each time an input slot has received a signal [0].

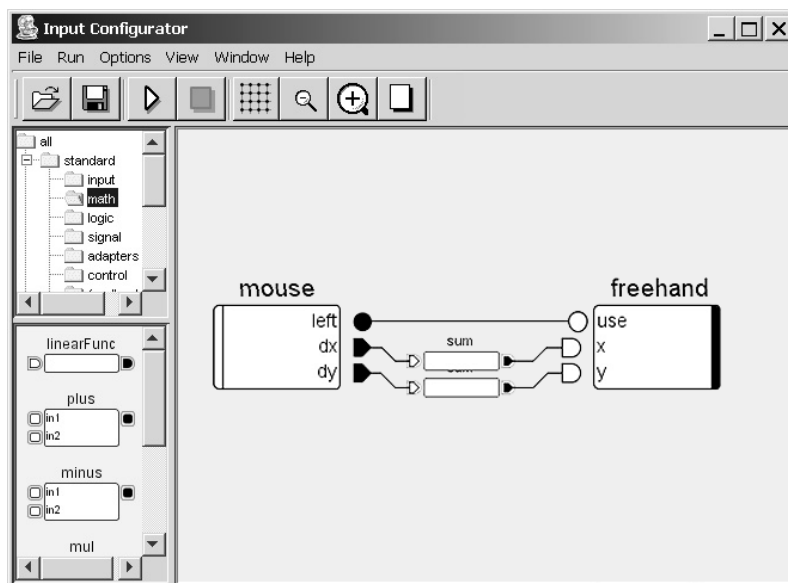


Fig. 3. A screenshot of the Input Editor.

The Input Editor

ICON configurations can be built or modified by direct manipulation through a graphical editor. An early prototype of this editor has been described in [0]. In this

contribution, the authors showed how the behavior of a standard mouse/keyboard configuration could be easily changed using the editor and its dedicated interaction techniques. In [0], we also give a subset of interaction techniques that can be described with our graphical notation and directly built using ICON.

The Fig. 3 shows a screenshot of the Input Editor window. Library devices and available system and application devices are listed on the left pane, and organized in folders just like a file system. Clicking on a folder (top left pane) displays the devices it contains (bottom left pane). Those devices are dragged on the editing pane to be used. The minimalist input configuration shown on the editing pane of the Figure 7 describes how a freehand tool from a drawing application called ICONDraw [0] is controlled using the mouse. The “sum” devices convert relative (delta) positional values sent by the low-level mouse into absolute values.

The toolbar on the top of the window contains two buttons for executing and stopping the input configuration. Execution is fast and does not need compilation, thus allowing easy testing and refinement of input configurations.

One simple example: One-Handed and Two-Handed Rubber Banding

ICON’s graphical editor allows the application designer to quickly build and test input configurations that make use of alternative sets of physical input devices, or modify existing configurations to adapt to enriched or impoverished input. Fig. 4 illustrates how a conventional technique can be changed into a Post-WIMP technique when a new input device (a graphical tablet) becomes available. The left upper part of the Fig. 4 shows the part of ICONDraw’s default input configuration which describes the standard rubber-banding technique for drawing lines: the user indicates the first end of the segment by pressing the mouse button, then the other end by dragging and releasing the button. The “firstThen” device encapsulates the simple automaton which implements this behavior. As shown on the lower part of the Fig. 4, this configuration has then been simplified so that each end of a segment being created is controlled by a separate pointing device. By doing this, the designer has just described a very basic bimanual interaction technique (Figure 8 on the right).

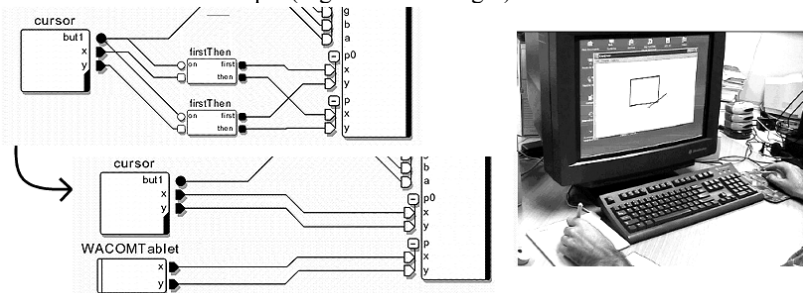


Fig. 4. A conventional line drawing technique, modified to make use of a second pointing device.

Dialogue Modelling and Prototyping

This section recalls the main features of the ICO formalism, which we use to model the case study. We encourage the interested reader should look at [0, 0] for a complete presentation of the formal description technique.

Overview of the ICO formalism

The Interactive Cooperative Objects (ICOs) formalism is a formal description technique dedicated to the specification of interactive systems [0]. It uses concepts borrowed from the object-oriented approach to describe the structural or static aspects of systems, and uses high-level Petri nets [0] to describe their dynamic or behavioural aspects.

Petri Nets is a graphical formalism made up of four components: the state variables (called place, depicted as ellipses), states changing operators (called transitions, depicted as rectangles), arcs, and tokens. Tokens are held by places; arcs link transitions to places and places to transitions. The current state of a system is fully defined by the marking of the net (i.e., both the distribution and the value of the tokens in the places). For a state change to occur a transition must be fired. A transition is fireable if and only if each of its input places holds at least one token. When the transition is fired, one token is removed from each input place and a token is deposited in each output place.

ICOs are dedicated to the modelling and the implementation of event-driven interfaces, using several communicating objects to model the system, where both behaviour of objects and communication protocol between objects are described by Petri nets. The formalism made up with both the description technique for the communicating objects and the communication protocol is called the Cooperative Objects formalism (CO and its extension to CORBA COCE [0]).

In the ICO formalism, an object is an entity featuring four components:

Cooperative Object (CO): a cooperative object models the behaviour of an ICO. It states how the object reacts to external stimuli according to its inner state. This behaviour, called the Object Control Structure (ObCS) is described by means of high-level Petri net. A CO offers two kinds of services to its environment. The first one, described with CORBA-IDL [0], concerns the *services* (in the programming language terminology) offered to other objects in the environment. The second one, called *user services*, provides a description of the elementary actions offered to a user, but for which availability depends on the internal state of the cooperative object (this state is represented by the distribution and the value of the tokens (called marking) in the places of the ObCS).

Presentation part: the Presentation of an object states its external appearance. This Presentation is a structured set of widgets organized in a set of windows. Each widget may be a way to interact with the interactive system (user → system interaction) and/or a way to display information from this interactive system (system → user interaction).

Activation function: the user → system interaction (inputs) only takes place through widgets. Each user action on a widget may trigger one of the ICO's user

services. The relation between user services and widgets is fully stated by the activation function that associates to each couple (widget, user action) the user service to be triggered.

Rendering function: the system \rightarrow user interaction (outputs) aims at presenting to the user the state changes that occurs in the system. The rendering function maintains the consistency between the internal state of the system and its external appearance by reflecting system states changes.

ICO are used to provide a formal description of the dynamic behaviour of an interactive application. An ICO specification fully describes the potential interactions that users may have with the application. The specification encompasses both the "input" aspects of the interaction (i.e., how user actions impact on the inner state of the application, and which actions are enabled at any given time) and its "output" aspects (i.e., when and how the application displays information relevant to the user). Time-out transitions are special transitions that do not belong to the categories above. They are associated with a timer that automatically triggers the transition when a dedicated amount of time has elapsed. When included in a system model such transition is considered as a system transition. They can also be included in a user model representing spontaneous user's activity.

An ICO specification is fully executable, which gives the possibility to prototype and test an application before it is fully implemented [0]. The specification can also be validated using analysis and proof tools developed within the Petri nets community and extended in order to take into account the specificities of the Petri net dialect used in the ICO formal description technique.

ICO Models for a rubber banding interaction technique

The *rubber banding* is a very classical interaction technique used in most graphical tools. It allows a user to draw a line (or a shape) based on the "drag and drop" interaction technique, where, while dragging, a temporary line is drawn, called ghost. We present here, through this classical example, the four parts of an ICO specification: the behaviour, the presentation part and the link between them stated by the activation and the rendering function.

1. Behaviour (ObCS). The behaviour of the rubber banding application is represented by its ObCS shown in Fig. 5. Initially, the application is in an idle state. When the mouse button is pressed, it starts the drawing of a ghost that is updated while moving the mouse pointer (dragging). When the mouse button is released, the definitive line is drawn, and the application returns in its idle state.

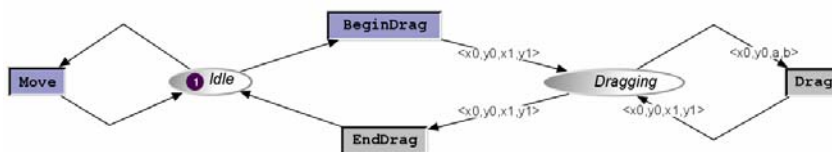


Fig. 5. Behaviour of the rubber banding interaction technique

- 2. Presentation part.** The presentation part described the external presentation part of the drawing line application. We describe hereafter (Fig. 6) a set of basic rendering methods that characterise the DrawablePanel. This set of methods is used to produce rendering by the rendering function (see the point 3).

```

Class DrawableJPanel
  Rendering methods {
    drawGhost(int x0, int y0, int x1, int y1) {
      //Draw a dashed line between point (x0, y0)
      //and point (x1, y1).
    }
    eraseGhost(int x0, int y0, int x1, int y1) {
      //Erase the dashed line drawn between
      // point (x0, y0) and point (x1, y1).
    }
    drawLine(int x0, int y0, int x1, int y1) {
      //Draw a line between point (x0, y0)
      //and point (x1, y1).
    }
  }
}
    
```

Fig. 6. Overview of the widget implied in the rubber banding application.

- 3. Rendering Function.** The rendering function describes how state changes impact the presentation part of the application. As state changes are linked to token movements, rendering items may be linked to either place or transition. Figure 7 describes the rendering function for the rubber banding application. The first line, for instance, shows that when a token enters the place Dragging, the corresponding rendering is to draw a ghost between the coordinates brought by the token.

ObCS element		Rendering method
Name	Feature	
Place Dragging	Token <x0, y0, x1, y1> Entered	drawGhost(x0, y0, x1, y1)
	Token <x0, y0, x1, y1> Removed	eraseGhost(x0, y0, x1, y1)
Transition EndDrag	Fired with <x0, y0, x1, y1>	drawLine(x0, y0, x1, y1)

Fig. 7. Rendering function of the rubber banding application.

- 4. Activation Function.** The activation function (shown by Fig. 8) relates the events produced by a widget to the transitions of the ObCS. Thus if the transition is fireable and the event is produced (by a corresponding user action on the widget) then the transition is fired (and its action is executed).

Widget	Event	Service
--------	-------	---------

Panel	Move	Move
Panel	MouseDown <x, y>	BeginDrag
Panel	MouseDrag <x, y>	Drag
Panel	MouseReleased <x, y>	EndDrag

Fig. 8. Activation function of the rubber banding application

Overview of PetShop Environment

In this section we present precisely how PetShop environment supports the design process of interactive systems. Some screen shots are included in order to show what is currently available.

ObCS Editor

Our approach is supported by a tool call PetShop which includes a distributed implementation of high-level Petri net interpreter written in Java. All the components of the ObCS can be directly built using PetShop. PetShop also automatically generates an Object Petri net from the IDL description [0]. The edition of the Object Petri net is done graphically using a palette of tools. The left part of the toolbar is used for generic functions such as load, save, cut copy and paste. The right hand side of the toolbar drives the execution of the specification.

Edition of the Presentation

Currently, PetShop is linked to JBuilder environment for the creation of the presentation part of the ICOs. Thus creation of widgets is done by means of JBuilder interface builder. However, we have not yet created a visual tool for editing the rendering and the activation function that still have to be typed-in in Java.

Execution environment

A well-known advantage of Petri nets is their executability. This is highly beneficial to our approach, since as soon as a behavioural specification is provided in term of ObCS, this specification can be executed to provide additional insights on the possible evolutions of the system.

Fig. 20 shows the execution of the specification of the line drawing application in Petshop. The ICO specification is embedded at run time according to the interpreted execution of the ICO. At run time user can both look at the specification and the running application. They are in two different windows overlapping as in Fig. 20. The window Line Drawing Application corresponds to the execution of the window with the ICO model underneath. In this window we can see the set of transition that are currently fireable (represented in dark grey and the other ones in light grey). This is automatically calculated from the current marking of the Object Petri net. Each time the user acts in the Line Drawing Application windows, the event is passed on to the interpreter. If the corresponding transition is fireable then the interpreter fires it, performs its action (if any), changes the marking of the input and output places and performs the rendering associated (if any).

Coupling Input Configurations and Dialogue

This section presents how the two approaches have been effectively integrated. We show first how this coupling takes place at the model level (ICoM and ICOs) and then at the environment level (ICoN and PetShop).

Models Coupling: ICoM and ICOs

Whereas ICO's activation function lists the couples *Widget × Event* and the user services they trigger, ICoM describes how each event is produced. For space reasons we only present here a simplified integration between ICO and ICoM models.

In an ICO specification, the *Widget × Event* represents the higher level event triggered by a widget translating the classical input events it receives. A widget thus behaves as a transducer that converts lower level events into higher level events, called widget events.

A simple way to couple ICoM and ICO is to extend standard widgets in order to represent them as output devices in ICoM model. Thus the ICoM model describes the events needed by the widgets. These ICoM output devices are then connected to ICoM Input devices through links and via other bricks. The resulting ICoM configuration represents how user actions on the input devices feed the widget with the correct events.

For instance, the previous section describes the rubber-banding application, specified with ICO. The activation function (see Figure 7) shows the events produced by our DrawableJPanel widget (MouseMove, MouseDragged ...), but does not make explicit the input device(s) used. Even if, in this example, the use of a simple mouse seems natural, we want to be able to deal with other input devices (such as graphical tablet, joystick, motion capture ...). The DrawableJPanel needs three information ((x, y) coordinates and a dragging trigger) to produce the relevant higher level events. The corresponding ICoM device is presented by Fig. 9.

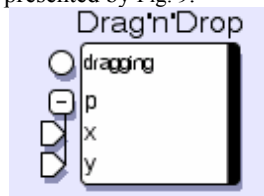


Fig. 9. ICoM output device representing inputs needed by the DrawableJPanel

Fig. 10 represents an ICoM configuration providing modelling the transformation of low level events on the mouse to transformed events in the output device.

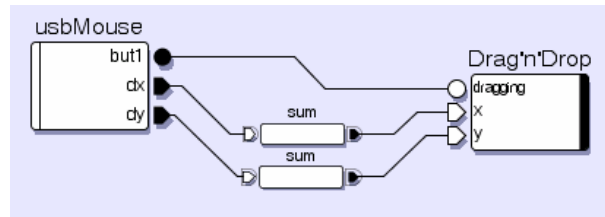


Fig. 10. ICoM model for DrawableJPanel

Systems Coupling: ICON and PetShop

In order to implement the link presented at the level of models in previous section, we need to make an application running within Petshop visible to ICON. This means that the set of widgets composing the presentation part, the activation and rendering functions and the dialogue part must register output devices as described above.

Initially, these applications are launched from the PetShop environment. While running, an input configuration can be deactivated using the Alt-C keystroke. This is essential as ICON allows redefining input handling at a very low-level, which can possibly hang all the system. For similar reasons, input configurations can be edited while paused but not while running. In contrast, the edition and simulation of the ICO model within Petshop is fully dynamic.

Case Study of a two handed line drawing application

In order to present the tool suite that we have developed for the engineering and very-high prototyping of multimodal interactive systems, this section presents the use of this tool suite on a case study. We first present the case study offering standard interaction technique and show how this case study can be easily extended in order to be manipulated by means of various input devices and thus using multimodal interaction techniques.

The line drawing application

This application (shown on Fig. 11) allows a user to handle a line, defined by two points. Modification of the line uses a rubber banding-like interaction technique for each point.

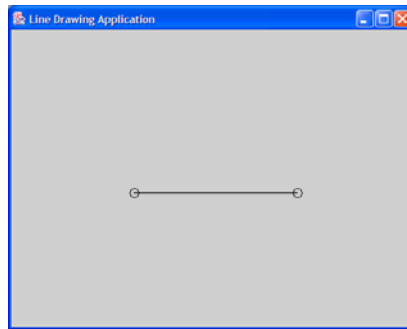


Fig. 11. The line drawing application

Application specification

Behaviour (ObCS). The ICO model in Fig. 12 describes the behaviour of the rubber banding interaction technique. Initially, the application is in an idle state. When the mouse button is pressed on the left point (resp. right point), it starts the drawing of a ghost (a dashed line). While moving the mouse pointer (dragging) the dashed-line is updated. When the mouse button is released, the definitive line is drawn, and the application returns in its idle state. With respect to the rubber banding interaction technique presented in Fig. 5 the model is duplicated here as two rubber banding are available at a time (one for each end of the line).

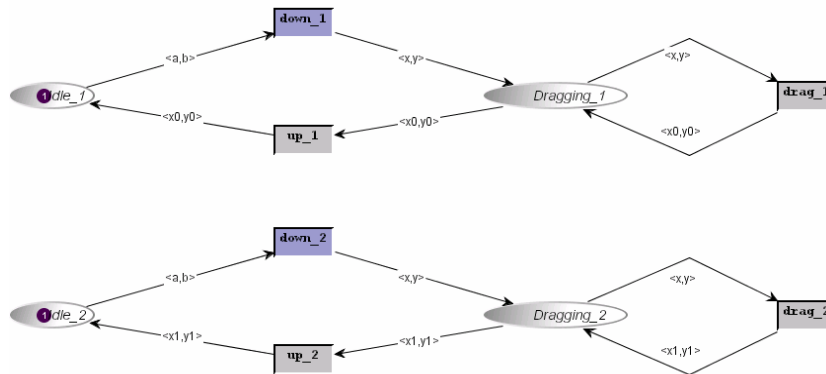


Fig. 12. Behaviour of the line drawing application.

Presentation part. The presentation part describes the external presentation part of the application. We describe hereafter (Fig. 13) a set of basic rendering methods that characterise the LineDrawingJPanel. This set of methods is used to produce rendering by the rendering function described in next section.

```

Class LineDrawingJPanel
  Rendering methods {
    drawGhost1(int x, int y) {
      //Draw a dashed line between point (x, y)
      //and the second point of the line.
    }
    eraseGhost1(int x, int y) {
      //erase the dashed line between point (x, y)
      //and the second point of the line.
    }
    drawLine1(int x, int y) {
      //Draw a line between point (x, y)
      //and the second point of the line.
    }
    drawGhost2(int x, int y) {
      //Draw a dashed line between point (x, y)
      //and the first point of the line.
    }
    eraseGhost2(int x, int y) {
      //erase the dashed line between point (x, y)
      //and the first point of the line.
    }
    drawLine2(int x, int y) {
      //Draw a line between point (x, y)
      //and the first point of the line.
    }
  }
}

```

Fig. 13. Overview of the widgets employed in the line drawing application.

Rendering Function. The rendering function describes how state changes in the Petri net describing the behaviour of the application impact the presentation part of the application. As state changes are linked to token moving from places to places, rendering items may be linked to either place or transition. Fig. 14 describes the rendering function for the drawing line application. The first line, for instance, shows that when a token enters the place Dragging, the corresponding rendering is to draw a ghost between the coordinates brought by the token.

ObCS element		Rendering method
Name	Feature	
Place Dragging_1	Token <x, y> Entered	drawGhost1(x, y)
	Token <x, y> Removed	eraseGhost1(x, y)
Transition Up_1	Fired with <x, y>	drawLine1(x, y)
Place Dragging_2	Token <x, y> Entered	drawGhost2(x, y)
	Token <x, y> Removed	eraseGhost2(x, y)
Transition Up_2	Fired with <x, y>	drawLine2(x, y)

Fig. 14. Rendering function of the line drawing application.

Activation Function. The activation function (shown by Fig. 15) relates the events produced by a widget to the transitions of the ObCS. Thus if the transition is fireable and the event is produced (by a corresponding user action on the widget) then the transition is fired (and its action is executed). The events produced are linked to one

of the two points of the line. MouseDown1, MouseDrag1 and MouseReleased1 represents classical drag'n'drop events that occurs related to the first point. The three others events are linked to the second point.

Widget	Event	Service
LineDrawingJPanel	MouseDown1 <x, y>	Down 1
LineDrawingJPanel	MouseDrag1 <x, y>	Drag 1
LineDrawingJPanel	MouseReleased1 <x, y>	Up 1
LineDrawingJPanel	MouseDown2 <x, y>	Down 2
LineDrawingJPanel	MouseDrag2 <x, y>	Drag 2
LineDrawingJPanel	MouseReleased2 <x, y>	Up 2

Fig. 15. Activation function of the line drawing application

Interface between the ICO specification and ICoM

As stated in section 4, the widget part is extended into an ICoM output device. Fig. 16 shows the ICoM model that represents the inputs needed by the line drawing application.

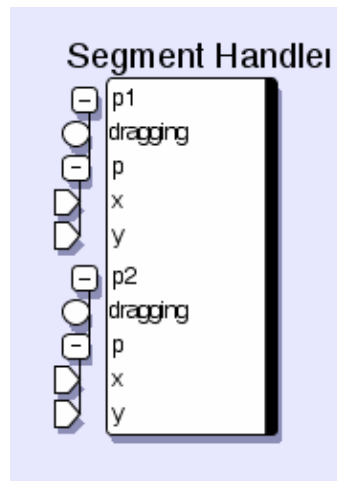


Fig. 16. ICoM device representing inputs needed by the LineDrawingJPanel of the ICO specification

Input configuration of the conventional line drawing application

The input configuration of the line drawing application describes how it is manipulated with a mouse. Fig. 17 shows this configuration: Mouse moves are transformed to coordinates (sum components) then used to animate a mouse cursor on top of the application frame (cursor component). In addition to the coordinates, the

cursor propagates also the state of the left mouse button to the rest of the configuration. *Shortcuts*, represented by grey vertical lines, are used to display the *same* cursor device at different places of the configuration (this means that the same cursor can manipulate both ends of the line).

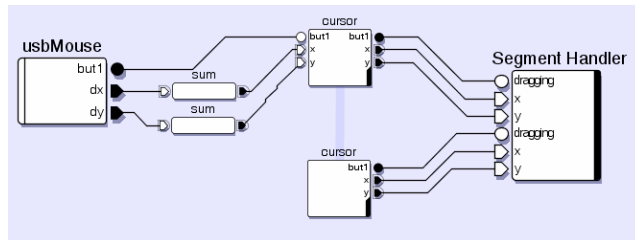


Fig. 17. Input configuration of the conventional (i.e. monomodal) line drawing application

The two copies of the cursor device thus provide the LineDrawingJPanel (of the ICO specification) with the correct parameters (i.e. x and y coordinates and the dragging state).

Two handed line drawing application

This section presents a modification of the case study in order to allow for two handed interaction on the line drawing application. The point is not here to discuss about the usability of such interaction technique but to show the impact of changing the behaviour of the application from monomodal interaction technique to a multimodal one and how the integrated approach proposed in this paper can deal with it.

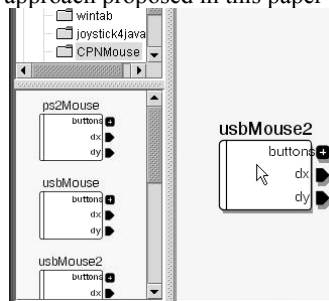


Fig. 18. A screenshot of ICON's editor with all available (connected) mice showing on the left pane (2 USB mice and a PS2 Mouse)

We describe a scenario in which the default input configuration is modified to handle two mice. In this scenario, each mouse moves a dedicated pointer but both pointers are used in the same way to control each extremity of the line. This allows both symmetric bimanual interaction and two-user collaborative interaction with the line.

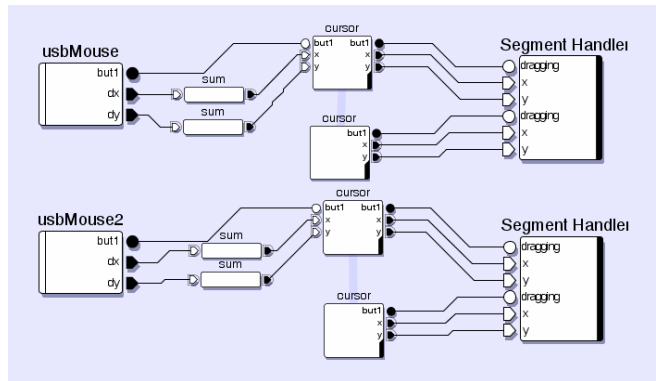


Fig. 19. Input configuration of the two-handed line drawing application

When launched, ICON’s editor also shows on the left pane all currently connected mice as *individual devices*, including PS/2, serial and USB mice (see Fig. 18). The user just has to identify the mice he wants to use (USB mice are sorted according to the HUB port they are connected to) and drag them in the edition pane. Note that other pointing devices such as graphical tablets can also be used, or even emulated with devices such as keyboard or voice recognition.

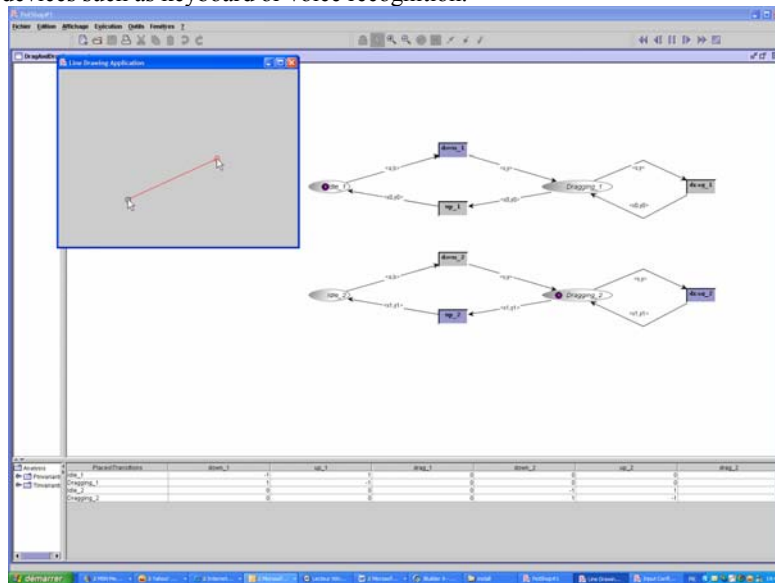


Fig. 20. Executing the two-handed drawing line application within PetShop

As both pointers share the same behaviour, the configuration described in Fig. 17 only has to be duplicated and mouse devices replaced. Lastly, two instances of this compound device are instantiated and connected to two separate USB mice, as shown on Fig. 19.

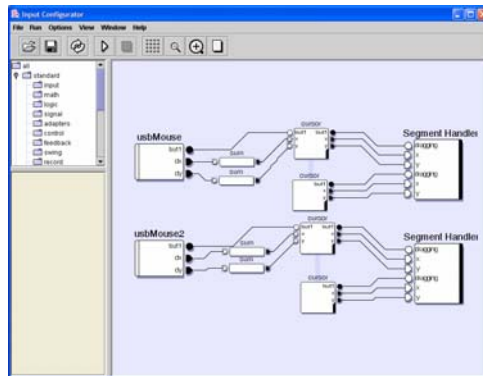


Fig. 21. Executing the two-handed drawing line application within ICON

When the configuration is edited, it may be executed. Fig. 20 shows the execution of the two-handed line drawing application within PetShop. Due to the locality principle of Petri nets (the firing of a transition only has impact on its input and output places) there is no change to make from the model in Fig. 12 to make the application usable in a multimodal way.

Fig. 21 shows ICoN environment. It is important to understand that both environments are used at the same time. This makes it possible to modify the input configuration (for instance changing the button used for selecting the end of the line) by changing the lines in the configuration. Behavioral description of the application can also be changed using PetShop.

Conclusion

This paper has presented a tool suite dedicated to the engineering of multimodal interactive systems. The ICO formalism deals with the functional core and the dialogue part of multimodal interactive systems. The ICON notation deals explicitly with input devices and input configurations. As these two models are supported by dedicated edition, simulation and execution environments, we have shown how very high fidelity prototyping can be performed and its related impact at various levels of the Arch architectural model.

The application of the notations and tools has been shown on a simple case study i.e. a bimanual drawing interactive system. This simple case study has shown a precise example of each model as well as how their edition and simulation.

This work belongs to a more ambitious project (see acknowledgement section) dedicated to the engineering of multimodal interactive systems for safety critical applications including military aircraft cockpits and satellite ground stations. The aim of this work is not only to provide notations and tools for building multimodal interactive systems but also to support verification and validation in order to support certification activities that are a critical phase in the development process of interactive safety critical applications.

Acknowledgements

The work presented here is partly funded by French defence agency (Direction Générale pour l'Armement) under contract n° 00.70.624.00.470.75.96 and by the French Space Agency CNES (Centre National d'Etudes Spatiales) under the R&T action n°CC201*02. Special thanks are due to Didier Bazalgette for precise information about the field of command and control systems in military applications.

References

1. L. Bass, R. Little, R. Pellegrino, S. Reed, R. Seacord, S. Sheppard & M. R. Szezur. (1991) The Arch Model: Seeheim Revisited. *User Interface Developers' Workshop*. Version 1.0.
2. R. Bastide & P. Palanque. (1995) A Petri-Net Based Environment for the Design of Event-Driven Interfaces. *16th International Conference on Applications and Theory of Petri Nets, ICATPN'95*, Torino, Italy, 66-83. Giorgio De Michelis, and Michel Diaz, Volume editors. Lecture Notes in Computer Science, no. 935. Springer.
3. R. Bastide, P. Palanque, Le Duc H., and Muñoz J. Integrating Rendering Specifications into a Formalism for the Design of Interactive Systems. Proceedings of the 5th Eurographics workshop on Design, Specification and Verification of Interactive systems DSV-IS'98. 1998. Springer Verlag
4. R. Bastide, O. Sy, P. Palanque, and D. Navarre. Formal specification of CORBA services: experience and lessons learned. ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'2000); Minneapolis, Minnesota USA. ACM Press; 2000: 105-117. ACM SIGPLAN Notices. v. 35 (10)).
5. G. Berry. (1999) The Esterel v5 language primer. Technical report, april 1999. <http://www-sop.inria.fr/meije/esterel/doc/main-papers.html>.
6. J. Coutaz, Paterno F. , Faconti G. , and Nigay L. A Comparison of Approaches for Specifying MultiModal Interactive Systems. Proceedings of the ERCIM Workshop on Multimodal Human-Computer Interaction. 165-174. 1993.
7. P. Dragicevic & J-D. Fekete. (2001) Input Device Selection and Interaction Configuration with ICON. Proceedings of IHM-HCI 2001, Blandford, A.; Vanderdonckt, J.; Gray, P., (Eds.): People and Computers XV - Interaction without Frontiers, Lille, France, Springer Verlag, pp. 543-448.
8. P. Dragicevic & J-D. Fekete. (2002) ICON: Input Device Selection and Interaction Configuration. Companion proceedings of UIST'02, 15th Annual Symposium on User Interface Software and Technology, Paris, October 2002.
9. P. Dragicevic & J-D. Fekete. (2004) ICON: Towards High Input Adaptability of Interactive Applications. Internal Report 04/01/INFO, Ecole des Mines de Nantes. Nantes, France.
10. O. Esteban, S. Chatty, and P. Palanque. Whizz'Ed: a visual environment for building highly interactive interfaces. Proceedings of the Interact'95 conference, 121-126. 1995.
11. H. J. Genrich. Predicate/Transition Nets, in K. Jensen and G. Rozenberg (Eds.), High-Level Petri Nets: Theory and Application. Springer Verlag, Berlin, pp. 3-43.
12. N. Halbwachs, P. Caspi, P. Raymond, D. Pilaud. (1991) The synchronous data-flow programming language LUSTRE. In Proceedings of the IEEE, volume 79, September 1991.

13. Hinckley, K., Czerwinski, M., Sinclair, M., Interaction and Modeling Techniques for Desktop Two-Handed Input, ACM UIST'98 Symposium on User Interface Software & Technology, pp. 49-58.
14. D. Navarre, P. Palanque, R. Bastide & O. Sy. Structuring Interactive Systems Specifications for Executability and Prototypability. 7th Eurographics Workshop on Design, Specification and Verification of Interactive Systems, DSV-IS'2000, Limerick, Ireland, 2000, Lecture notes in Computer Science n° 1946.
15. OMG. The Common Object Request Broker: Architecture and Specification. CORBA IIOP 2.2 /98-02-01, Framingham, MA (1998).
16. P. Palanque & A. Schyn. A Model-Based Approach for Engineering Multimodal Interactive Systems in INTERACT 2003, IFIP TC 13 conference on Human Computer Interaction.
17. R. Wieting 1996. Hybrid High-Level Nets . Page 848 855 Proceedings of the 1996 Winter Simulation Conference. ACM Press.
18. J.S. Willans & Harrison M. D. Prototyping pre-implementation designs of virtual environment behaviour. 8th IFIP Working conference on engineering for human-computer interaction (EHCI'01) 2001. LNCS, Springer Verlag.

Discussion

[Rick Kazman] The context of this is safety critical systems. Two properties to address are reliability and performance. How do you guarantee that in the model you are presenting that these properties are there and, given that the model is compositional, that the properties are preserved?

[Philippe Palanque] The intention is not to embed PetShop in an aircraft. The model is intended to be a specification and a high-fidelity prototype. So we produce a specification and a running example. On the aeroplane, for example, it was necessary to have response within 20ms. This is met with our system. We hope to provide a set of tests as well to allow the developers to be sure that they have met the requirements. We are working on this now.

[Bonnie John] In the spirit of the grand challenge of the "UI crash test dummy", have you thought of attaching this to a cognitive modeling architecture such as ACT-R (which has its own model of human-like concurrency and human-scale timing?)

[Philippe Palanque] We work at a low level. So we use Fitts' Law for example, to tell us that the average time for a user to respond will be some value. Petri Nets allow time to be attributed to arcs and specification of the size of buttons, which allow this kind of analysis.

[Michael Harrison] Petri nets have a lot of "good" baggage allowing you to prove many properties of systems. You presented this tool primarily as a rapid prototyping environment. Have you taken advantage of the properties of Petri nets for analysis?

[Philippe Palanque] There is a tradeoff in designing Petri nets for evaluation vs prototyping. In the past we've worked on the modelling approach, but now we're looking at expressiveness. We have performed analyses such as invariant checking.

[Michael Harrison] Do you feel this is a good way of specifying this kind of system?

[Philippe Palanque] We have a contract with the French certification authority. They have no idea of how to certify a cockpit. Now several people at Thalès are using our tools to work on this.

[Willem-Paul Brinkman] Synchronization over feedback is also important as well as synchronization of inputs. Do you handle this?

[Philippe Palanque] Our approach can handle the specification of the entire system. We have seen this in practice. For example, in the A380, they have a server (X Windows). There is feedback indicating that the server has not yet received feedback from the application, during which the pilot must wait.

[Grigori Evreinov] There was no clear definition of multi-modal. What is the difference between multi-modal and multi-channel interaction? E.g., if you can manipulate with two mice, it's two channel manipulation. If you have speech fused with mouse motion, it's multi-modal. Content should not be fused in head of the user.

[Philippe Palanque] You are right. The example was multi-channel interaction. The point was to show the integration of multiple devices. For multi-modal, we can have models of two mice, which are fused via a single model at the logical interaction level. This is perfectly possible with PetShop. For example, using two fingers on a touch-sensitive display.