Published in Proceedings of Eurographics Workshop DSV-IS'96

A Formal Description of Low Level Interaction and its Application to Multimodal Interactive Systems

Johnny Accot Stéphane Chatty Philippe Palanque

Centre d'Études de la Navigation Aérienne
7 avenue Edouard Belin
31055 TOULOUSE CEDEX, FRANCE
D.G.P.
University of Toronto
Toronto, ON M5S 1A4, CANADA
LIS - IHM
Université Toulouse 1
31042 TOULOUSE CEDEX, FRANCE
{accot,chatty,palanque}@cena.dgac.fr

Abstract

The lack of formal models for describing low-level interaction restricts programmers to interactors provided by toolkits. It impedes the construction of highly interactive systems and the design of new interaction styles, such as multimodal interaction. This article reports on our experience with formalising low-level graphical interaction. We propose primitives for event specification and handling that can be used along with Petri nets to model such interactions. We then show how multimodal interactions can be built from monomodal ones by combining those models. This is examplified by an experimental two-handed graphical editor that has been built using the proposed model.

Keywords

Formal specification, two-handed interaction, multimodal interfaces, Petri nets.

1. Introduction

Though this is an active field of research [Brun 95], most interactive systems designed today do not rely on formal development or specification methods. This is especially true as far as low-level interaction in user interfaces is concerned. This leads interface programmers to stick to widespread interaction styles and to rely on the experience of toolkit designers. Those limitations cause serious problems to the designers of complex and critical systems such as air traffic control displays: either they use well-known interaction styles, with possible consequences on the usability

of the system, or they are faced with the problem of specifying low level interaction without any help for that purpose. Another consequence of that absence of formal models is the difficulty to teach graphical interface construction. Whereas basic algorithms such as list sorting or rendezvous are described in numerous books, no formalism is available to describe even a double-click or the behaviour of a menu in a precise way. Building widgets is thus closer to craft knowledge than to a reliable and structured engineering process. This leads to unacceptable situations such as simple widgets of widespread commercial toolkits exhibiting incoherent behaviours.

This lack of models for graphical interaction has obvious consequences on the design of more complex systems. In multimodal interfaces, for instance, parallel event flows can lead to unpredictable incoherent situations in the same way as in multitasking concurrent programming systems. What is a nuisance with graphical interaction can become an obstacle with multimodal interaction. Our previous work on two-handed interaction [Chatty 94] confronted us with this problem, for instance when trying to build a two-handed graphical editor, using techniques such as Toolglass and Magic Lenses [Bier 94]. In order to be able to specify precisely and without ambiguities the expected behaviour of the system, we were faced with the need to use formal methods. This paper reports on the solutions we proposed and implemented to describe discrete aspects of traditional and multimodal interaction at a low level of interaction.

The next section proposes a flexible event specification scheme proposed as a replacement for callbacks. Section 3 shows how this specification scheme can be fruitfully integrated with Petri nets to fully describe event-based interactions. This leads to a generic framework for modelling low level interactions. Section 4 shows how that framework can be easily exploited to deal with multiple threads of input for multimodal interaction. Section 5 gives an example of how a model designed with that framework can be implemented in a real application. The example proposed here is a graphical editor with two-handed interaction capabilities. In the last section the work described in this paper is positioned relatively to previous work in the field of dialogue modelling.

2. Primitives for event description

When modelling interactive systems, the interaction primitives are at least as important as the formalism used to manipulate them, because they determine the level of control that programmers have over the behaviour of the system. In an event-based model, the most important primitives are the event selection scheme and the method of linking events to behaviours. Most toolkits select events by their types, the window in which they occur, and sometimes the graphical object located under the cursor. They use callbacks to associate events to behaviours: when selecting a class of events, the procedure to be called is specified. However, callbacks are known to be a problem, because of their low level of abstraction [Myers 90]. In addition, as shown in [Chatty 94], selection sometimes needs to use notions more precise than event types. For instance, one often needs to specify which button

of the mouse or which key of the keyboard was pressed. We thus considered an association of two primitives: *reactions* and *criteria*.

- A reaction links an object's method to a set of possible events. When an event matching the specification of the reaction occurs, the method is invoked. The object that reacts to the event does not have to be the graphical object on which the event occurred. It does not even have to be a graphical object. This allows us to have icons react to mouse movements on the background of a window (when dragging), or to have functional core objects react to key presses (when pressing 'q' to quit an application). A criterion is the basic entity used in the specification of reactions.

- Criteria check properties of events, and a reaction is triggered only if all its criteria are satisfied. With current callback-based models, event selection can be expressed as "bind this function to this type of events". With reactions and criteria, it can be expressed as "this object is interested in events having such and such properties". This allows designers to explicitly express constraints, instead of implementing them by hand in the callbacks. The most commonly used criteria are event types, event targets, and devices. Others can refer to the attributes of events (amount of movement, for instance) or to the state of any object. A classical example of amount of movement is the threshold parameter in the options of a mouse. The threshold helps users in performing clicks and double-clicks by ignoring small mouse movements between clicks.

Using criteria and reactions provides a flexible way of expressing simple behaviours of objects in an event-based application. However, more complex behaviours triggered by sequences of events need formalisms such as finite state machines or Petri nets to be expressed. This is the topic of the next section, where criteria are used to label the transitions of Petri nets.

3. Interaction level

This section aims at showing how it is possible (using the primitives for event description described in the previous section) to model the production of high level events from the physical model of an input device. We will show the model for a mouse and we will prove the consistency of the physical model with the one responsible for the generation of high level events (called interaction level events).

Interaction level correspond to the user's actions while manipulating physical devices such as a mouse or a keyboard. For example, such actions can be: press the right button on the mouse, release the left button of the mouse, press a key, etc. It can be easily seen that the user's behaviour heavily depends on the physical behaviour of the device, e.g. a button can only be released if it has been pressed before. Describing this behaviour is quite straightforward when dealing with classical devices such as mice and keyboards. The behaviour of a mouse featuring only one button is easily described with an automaton, or with a Petri net as shown in Figure 1. Petri nets are a formalism devoted to the modelling of discrete event

systems in which parallelism plays an important role. When modelling with Petri nets, a system is described in terms of state variables (called places and depicted as ellipses) and by state changing operators (called transitions and depicted as rectangles), connected by arcs. The state of the system is given by the marking of the net, which is a distribution of tokens in the net's places. This use of tokens allows designers to describe states in a very concise way.

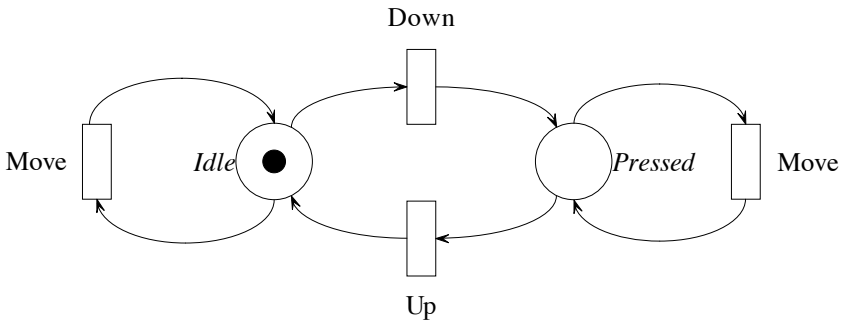

Figure 1. The physical behaviour of a one-button mouse

In Figure 1 the Petri net describes that, in the initial state, the device is idle: it waits for an event to occur. While in this state, two kinds of events may occur; this is represented by the two different outgoing transitions. The transition Move removes the token from the place and puts it back into it, which means that the system remains in the same state. If the event *Down* occurs, the token in the place *Idle* is removed and put in the place *Pressed* thus describing a change of state in the system. From that state, both Move and Up events may occur: Move keeps the system in the *Pressed* state, and Up puts it back to its initial state.

This model describes precisely the basic behaviour of the device. However, in order to fully exploit the benefits of this device, it is important to take into account a higher level of interaction represented by the events associated to sequences of physical actions. For example, it is important for the designer to be able to process the sequence (Down, Up) either as two different events or as a single event: Click. The management of those higher level events is less simple than that of the physical ones. For instance, double-clicks events involve a time constraint: the sequence (Down, Up, Down, Up) has to be performed within a given amount of time, else it is interpreted as two isolated clicks. This is usually implemented by activating a timer, and taking into account time-out events. In addition, if the mouse is moved during the sequence, this is not a double-click either. Figure 2 presents the different events associated to a mouse. The physical events are described in the left column, the interaction level ones in the right column.

In the case of mouse events, most applications are at least as much interested in higher level events as in the low level ones produced by the graphical layer. Clicks, drags and double-clicks are the results of sequences of physical events. Usually, these higher level events are produced either by dedicated algorithms, or through

d:	Button Down	C:	Single Click
u:	Button Up	DC:	Double Click
m:	Mouse Move	B:	Begin Drag
t:	Time Out	D:	Drag
		E:	End Drag

Figure 2. Physical (left column) and interaction level events.

models such as Garnet's interactors [Myers 90]. We call them interaction level events, as they may be used as input to other processes that would use them as basic events. The creation of those interaction level events depends on the state of the interaction, which in turn depends on the physical events that previously occurred. This type of behaviour is easily described with automata or Petri nets. We chose to use Petri nets, so as to be able to express parallelism and synchronisation, as we will see in the next sections. Using this formalism we have been able to test different designs for double clicks, and provided a good basis for thinking about interaction styles involving more than one device. It also proved useful when integrating exotic devices such as a telephone into graphical applications [Chatty 96a].

However, the use of basic Petri nets does not permit the description of the types of events and to change the behaviour of the model according to the type of these events. Thus we upgraded the basic model by adding the criteria and reactions presented in section 2. The next section shows how this integration can be used for describing high level event production.

3.1. An example of generation of interaction level events

Figure 3 shows the physical events consumed by the Petri net and the interaction level events it produces. This is one of the possible designs for clicks, double-clicks and drags according to both physical and higher level events. Labels associated with transitions have the following shape:

$$\frac{Event,Criteria/Action}{Production}$$

where

- Event is the low-level event consumed
- Criteria is the set of additional criteria that the event must satisfy for the state change to occur
- Production is a set of interaction level events that are produced during the state changing
- Action is the additional action performed by the model during the state change

Figure 3:

This Petri net describes the interaction level events policy for a given interaction style. It tells when and how those events are produced according to the user's actions on the device. In this Petri net, the policy works like a transducer: each time a physical event is accepted, the Petri net fires a transition and creates higher

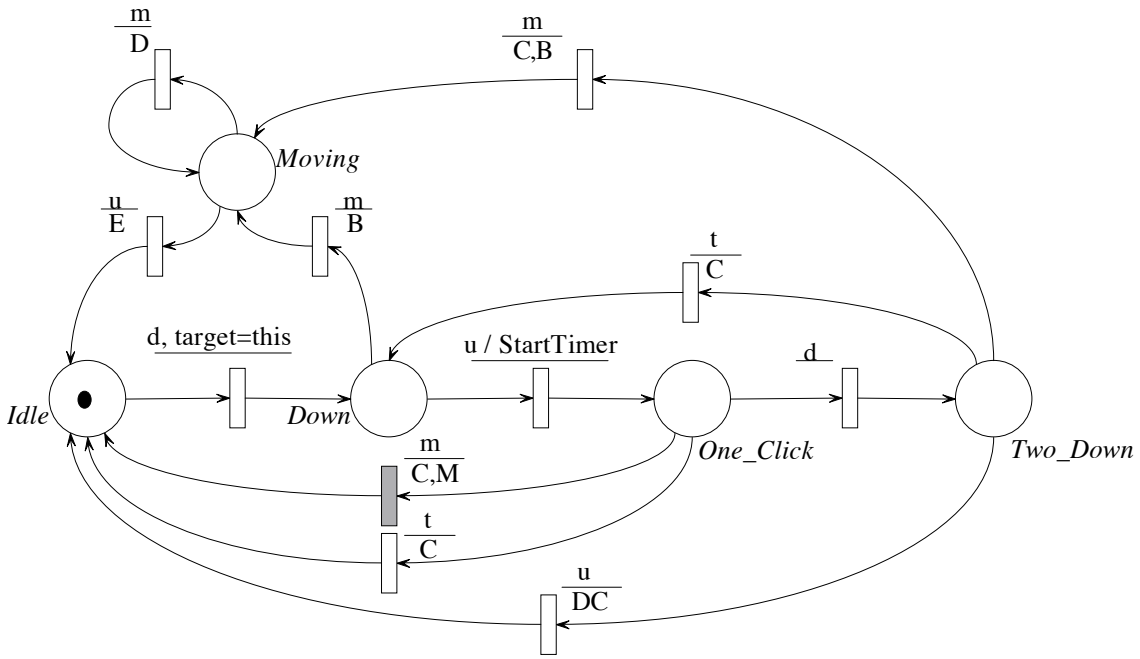

Figure 3. A design for clicks, double-clicks and drags

level events. For example, in the policy represented in Figure 3, the physical mouse-move (m) is transformed into a higher level mouse-move (M), e.g. the transition between places *One_Click* and *Idle* reacts to the event m by generating another event M plus an event click (C). However, all physical events are not immediately translated into interaction level events. For example, each event d received while the system is in the initial state, is consumed without any production.

In the simple model described in Figure 3, criteria are only event types, except for the first down event (the transition between places *Idle* and *Down*). More sophisticated designs could be obtained, for instance by attaching a criterion to the arc from place *Down* to place Moving so that small moves are ignored. The reaction StartTimer on the transition between place *Down* and place Click states that a timer is started as soon as the physical event up (`u`) is received.

The Petri net in Figure 3 was built as follows. The four states *Idle*, *Down*, *One_Click* and *Two_Down* and the transitions between them represent the sequence of physical events (`d, u, d, u`) that corresponds to a double click (`DC`). The timing constraint is represented by the action StartTimer on the transition between *Down* and *One_Click*, and by the transitions labelled by `t` on other states. The most significant such transition is the one that links *One_Click* to *Idle*: after a certain time; the sequence (`u, d`) is considered as a `C`. Similarly, the no-move constraint is expressed by the transitions that lead to place Moving. When in that state, all moves (`m`) are parts of a drag (`D`). A `m` received when in state *One_Click* also forces

the emission of a `C`, and the return to *Idle*, though the greyed out [1] transition in Figure 3.

3.2. Analysis of conformance between physical and interaction level events

An important point while designing a model for interaction level events is to make sure that the model is consistent with the underlying physical behaviour of the device. One possible way to prove this kind of property is to consider the two models as a client and a server cooperating together. In our case, the model describing higher level events (called MHLE) is the client and the one describing the physical events (called MPLE) the server. Thus, the low level events in the MHLE describe the demand of the model towards the MPLE. Conversely, the events modelled in the MPLE describe its offer.

Proving that the models are consistent is equivalent to proving that the demand of the client is included in the offer of the server. In previous work on the verification of the consistency of models in CSCW applications [Palanque 95a], we have shown that, for Petri nets, the demand and offer of the models are the same as the demand and offer of the automata corresponding to the marking graph of the Petri nets. Here, the Petri nets in Figure 1 and Figure 3 are state machines, hence their marking graphs are the models themselves. We thus only have to prove that the language of the automaton corresponding to MPLE is included in the language of the automaton corresponding to the MHLE.

Using regular expressions to represent the languages, the language accepted by the Petri net in Figure 1 is: `(m*dm*u)*` and the language accepted by the Petri net in Figure 3 is: `(dm*u | dum | dudmm*u | dudu)*`. The time-out events have not been taken into account in the calculation of the language because they can be considered as internal events of the MHLE. A more precise calculation could have been made by considering that those events are produced by another server responsible for handling temporal aspects.

It can easily be proved that the language generated by each of the terms of the second regular expression are included in the language generated by the first regular expression. Hence, the language generated by the second regular expression is included in the language generated by the first one. That proves that the MHLE only describes actions that can result from sequences of events emitted by the MPLE. In other terms, this means that the model of interaction level production only reacts to sequences of physical events that can be provided by the physical device. It is important to notice that the model is not supposed to react to all the possible sequences of the physical device as this is a matter of design.

The next section is devoted to show how the work presented above can be reused in order to model the production of multimodal events in an interactive system. The emphasis is not on the design process which will be the focus of a future paper but on the solution of the precise problem of generating multimodal events in two-handed multimodal systems.

---

[1] The is no special meaning for the greyed out transitions. It is only for the sake of readability thaht they have been greyed out.

4. Multimodal events

The technique described above can be used to describe multimodal interaction, as we will show here for the case of two- handed interaction. As shown in [Chatty 94], two-handed interaction is - at least technically speaking - a form of multimodal interaction, and exhibits the same variations. There are many possible two-handed interactions styles, and thus tools are needed to design, implement and evaluate them. Examples of such interaction styles are those that would involve touch screens manipulated with two or more fingers . With such devices, one can perform combined clicks (i.e. two fingers 'clicking' at the same time) and many other combinations. For instance, air traffic controllers currently use their finger tips to evaluate and compare distances on their radar screens. With touch-screen based displays such as those explored at CENA [Chatty 96b], such comparisons could be improved by detecting combined clicks and displaying the appropriate information. But such combinations are complex to implement, and we soon found that a formal method was necessary in order to handle all possible cases, and even to communicate among ourselves. We thus generalised the use of the framework described in the previous section. Here the set of multimodal events is limited to two: a combined click (a classical click done with both hands simultaneously) and a double combined click (a classical double click done with both hands simultaneously). However, the approach presented aims at being much more generic as we are currently working on composition techniques in order to describe other multimodal interaction level events, and to build them for other input devices.

The use of Petri nets allows us to describe two-handed interaction as an extension to traditional interaction. The presence of two similar devices is represented by the presence of two tokens in the same Petri net: this technique is known as folding and is the basis of an early extension to basic Petri nets called coloured Petri nets [Jensen 81]. For that reason, in Figure 4 the two pointers (possibly fingers) are represented by a grey and a black token. Places and transitions are added to model the interaction level events produced in reaction to combined actions on the devices. In the case of combined clicks, the extension to the model of Figure 3 is represented in the greyed out region of Figure 4. Two places were added and the transitions CC and CDC (which feature double incoming arcs and double outgoing arcs) detect the occurrences of combined clicks and combined double-clicks. For instance, a combined click occurs when two tokens reach place *Comb_Click* at nearly the same time.

The Petri net in Figure 4 is currently in the state when the user has pressed and released each pointer. This is described by two tokens in the place *Comb_Click*, thus enabling the transition with two arcs that will produce the CC event. The need for the click events to be temporally close is represented by the other output transition of the place *Comb_Click*, which consumes tokens according to the duration of their stay in that place. These black transitions correspond to timed transitions, which means that after being enabled they wait for a given amount of time before being fired. This modelling has been made possible thanks to the easy quantitative time modelling in Petri nets. This kind of Petri net is known as timed Petri nets and

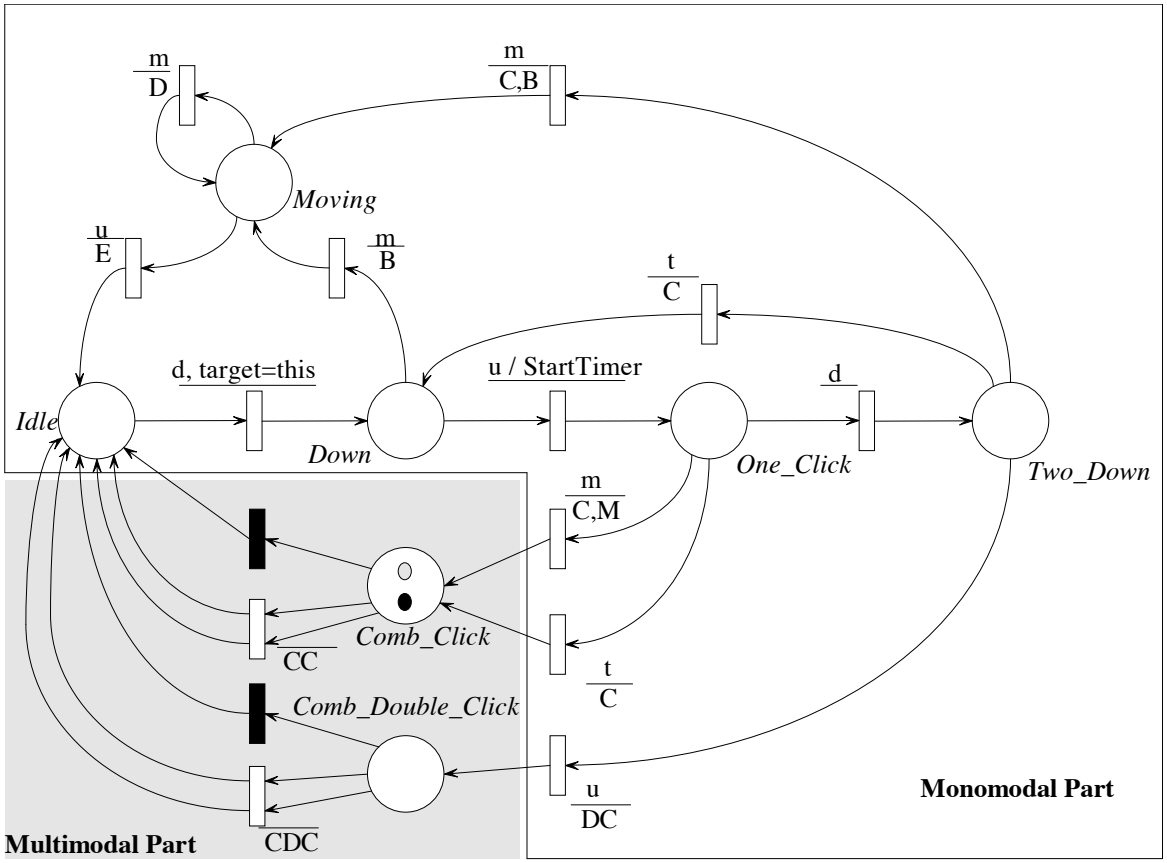

Figure 4. A Petri net managing two-handed interaction

further information about their functionalities can be found in [Palanque 95b].

As there is one token for each device it is important to be sure that an event occurring on a given device is associated to the corresponding token in the model. This is done by associating criteria to transitions. For example, at the initial state the user can press the button on each device. The criterion associated to the only available transition will select the token corresponding to the device currently used by the user and then the transition will remove the selected token from the place *Idle* and put it in the place *Down*. An example of easy representation of this kind of criteria is to use colours as the tokens have to be differentiated. However, as the production of CC and CDC events is related to quantitative temporal evolution of the model the expressive power of Coloured Petri nets is not sufficient.

5. Implementing an example: an editor for building automata

The aim of this section is to show on a simple example how the formal framework presented above can be used in real applications. The example we use is graphical editor for drawing automata, extended with two-handed input capabilities, that we developed as a test-bed for two-handed interaction styles. Though we are currently working on two-handed interaction from a usability point of view, we will here focus on very simple interaction techniques for the sake of clarity. We also provide more details on the implementation of our framework.

5.1. The example

The first step when designing an automaton consists in drawing the skeleton of the automaton. To do so, the editor provides users with two basic objects: states (represented by circles) and transitions (depicted by labelled arcs). It also features functions to manipulate these objects: add, move, delete, and edit.

The sample editor we have built is shown in Figure 5. It features a working area, where the automaton is built and displayed, and a set of buttons for three of the functions: create, move, and delete. Editing the contents of a transition is performed by double-clicking on its graphical representation, which opens a transition editor (not presented here). As an example of two-handed interaction, let us consider a cooperation of two hands (or rather two fingers). An example of such cooperation consists in the creation of a transition between two states by clicking on a state, then nearly simultaneously on another. In that case the transition is automatically created from the state that has been clicked first to the other one. The double click and the two-handed manipulations in this editor have been implemented using the Petri net model for the management of multimodal aspects presented in section 4 of this paper.

5.2. Implementation

Our implementation of the model is like traditional interpreted implementations of Petri nets [Feldbrudge 93] where arcs, places and transitions are stored in lists. Depending on the current marking, the only active criteria are those corresponding to transitions that might potentially be fired. Criteria are implemented through

an event subscription mechanism, so this means that the model will only receive events that are meaningful according to the current state.

Referring to Figure 4 for instance, if both tokens are present in the *Idle* place, the net is only sensitive to button-down events and does not receive any information regarding other kinds of events. Then, when one of the transition fires and the token moves from the *Idle* place to the *Down* place, the net unsubscribes to button-down events from the device corresponding to the token that moved, and subscribes to button-up and move events.

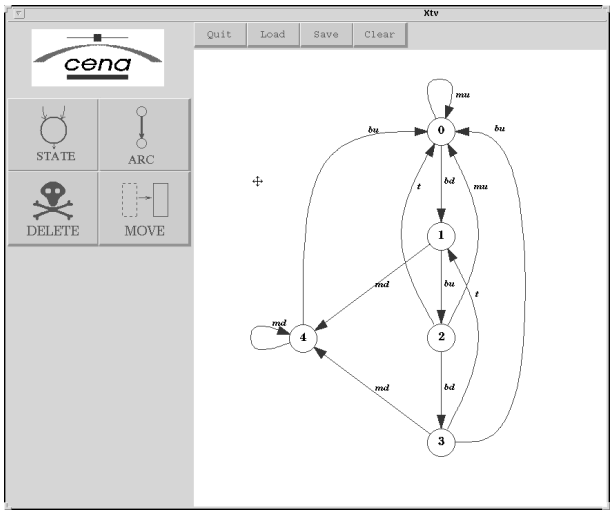

Figure 5. The editor for designing automata

This approach based on an interpretation of the Petri net a run-time has two important consequences on the characteristics of the net, regarding the choice and specialisation of behaviours:

- a graphical object is able to 'choose' a behaviour (by inheritance in our approach). In user interfaces for instance, a window does not have the same behaviour as a menu button: when a user grabs a window, this window should always be attached to the mouse pointer and all mouse drag events should be redirected to it, whereas a menu button should not grab all events event if the button-down was performed on it. A set of different "standard" behaviours would be proposed by graphical toolkits so that developers would just have to pick a behaviour and, if needed, modify it as suggested below.

- the Petri net can be 'customised', which means that developers or even end-users can easily change the interpretation of events from devices according to their needs. These modifications primarily concern the parameters of the net, such as time delays or movement tolerance if any, but also the net structure itself. Even if these dynamic changes are not fully understood, we believe that this may be possible and useful if assisting users in this task.

6. Related work

A lot of work has been devoted to the formal specification of interactive systems. This work has been done at different levels of abstraction from higher level such as sequencing of windows, sequencing of interaction objects triggering within a dialogue window, to elementary behaviour of a single interaction object. A large part of the results obtained have been applied to 'indirect manipulation interfaces', i.e. interfaces controlled through menus, buttons and dialogue boxes. Less research has been devoted to highly interactive interfaces featuring direct manipulation or animation.

Figure 6 presents a summary of the related work in the formal specification of interaction. This table does not represent more theoretical work on interactive systems such as [Duke 93, Harrison 90, Dix 91] in which the main point is to address properties of interactive systems rather than their inner behaviour.

The figure is organised as follows. Work located in the left column corresponds to work on indirect manipulation systems. Work located in the right column concerns highly interactive systems. The three rows correspond to the grain of dialogue taken into account. The lower row is related to the finest grain of dialogue (the behaviour of interaction objects themselves) while the upper one concerns higher grain of dialogue (such as the sequencing of local dialogues). The work presented in [Stasko 89] is in dashed lines because it only concerns animation and not interaction in applications. The work presented in this paper is located in the lower right box of Figure 6 as it concerns multimodal interactive systems with direct manipulation facilities.

7. Conclusion

We have shown in this paper how novel primitives for event description can be used as the basis of a formal model of low level interaction in direct manipulation interfaces. We have also shown how the same formalism, based on Petri nets, can be used to describe multimodal interaction in a way that allows the reuse of well known interaction styles to design new ones. This work has been integrated to the Xtv toolkit [Beaudouin-Lafon 90], and is currently used to experiment with the design and evaluation of two-handed interaction styles at the University of Toronto.

This works opens a whole set of new questions and perspectives. It suggests a more general event-based model of interactive applications, where physical events would successively be translated into more and more abstract events until they are caught by the functional core. In such a model, even the internal behaviour of widgets would be specified formally, thus allowing formal verification of the whole behaviour of an interface, and possibly performance evaluations within the GOMS methodology. This works also calls for its integration into a graphical direct manipulation interface editor such as Whizz'Ed [Esteban 95]. We are currently working on methods that would help the construction of complex behaviours from basic ones in such an editor. We are also working on the merging of this model with the data-flow model of Whizz [Chatty 92], so as to span as widely as possible

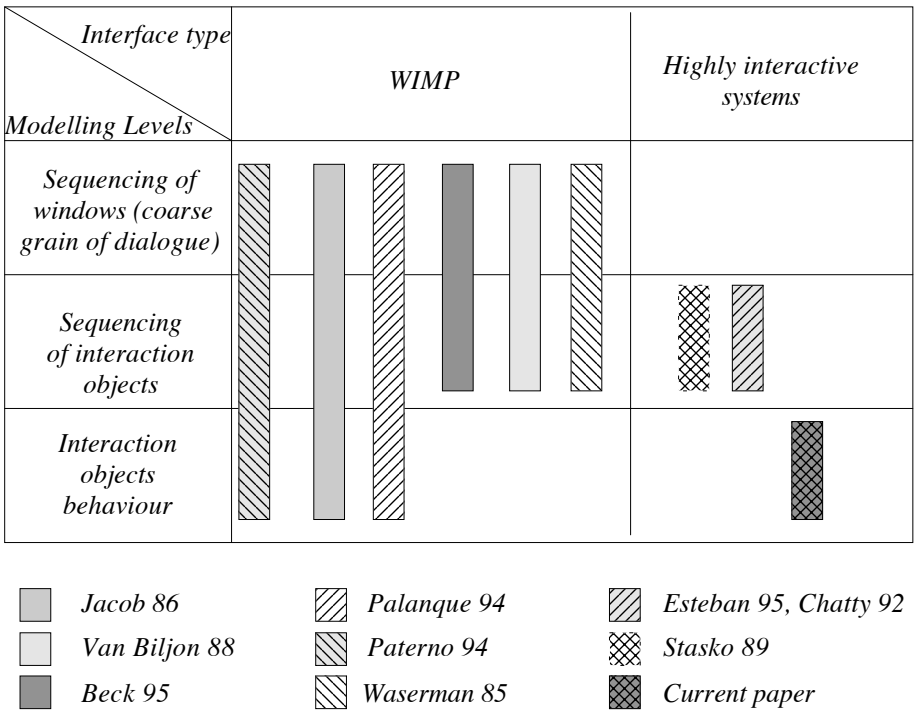

Figure 6. Relative position of related work on low level behaviour modelling

the spectrum of user interfaces behaviours, from simple callbacks to continuous phenomena such as animation.

8. REFERENCES

[Beaudouin-Lafon 90]
Michel Beaudouin-Lafon, Yves Berteaud, et Stéphane Chatty. Creating direct manipulation interfaces with X_{TV}. *Proceedings of EX'90, London*, pages 148–155, 1990.

[Bier 94]
Eric Bier, Maureen Stone, Ken Fishkin, William Buxton, et Thomas Baudel. A taxonomy of see-through tools. *Proceedings of the ACM CHI*, pages 358–364. Addison-Wesley, 1994.

[Brun 95]
Philippe Brun et Michel Beaudouin-Lafon. A taxonomy and evaluation of formalisms for the specification of interactive systems. *Proceedings of the Human-Computer Interaction (HCI'95)*, pages 197–212, 1995.

[Chatty 92]
Stéphane Chatty. Defining the behaviour of animated interfaces. *Proceedings of the IFIP WG 2.7 working conference*, pages 95–109. North-Holland, Août 1992.

[Chatty 94]
Stéphane Chatty. Extending a graphical toolkit for two-handed interaction. *Proceedings of the ACM UIST*, pages 195–204. Addison-Wesley, Novembre 1994.

[Chatty 96a]
Stéphane Chatty, Patrick Girard, et Stéphane Sire. Vers un support multimédia au collecticiel synchrone. *Technique et Science Informatique*, 15(9):–, 1996.

[Chatty 96b]
Stéphane Chatty et Patrick Lecoanet. A pen-based workstation for air traffic controllers. *Proceedings of the ACM CHI*. Addison-Wesley, 1996.

[Dix 91]
Alan Dix. *Formal Methods for Interactive Systems*. Academic Press, 1991.

[Duke 93]
David Duke et Michael Harrison. Abstract interaction objects. *Computer Graphics Forum, Proceedings of Eurographics 93*, volume 12, pages 25–36, 1993.

[Esteban 95]
Olivier Esteban, Stéphane Chatty, et Philippe Palanque. Whizz'Ed: a visual environment for building highly interactive software. *Proceedings of Interact'95*. Chapman and Hall, Juin 1995.

[Feldbrudge 93]
F. Feldbrudge. Petri nets tool overview 1992. G. Rozenberg, editor, *Advances in Petri nets, LNCS 674*, pages 169–209. Springer-Verlag, 1993.

[Harrison 90]
Michael Harrison et Harold Thimbleby. *Formal Methods in Human Computer Interaction*. Cambridge University Press, 1990.

[Jensen 81]
Kurt Jensen. Coloured petri nets and the invariant method. *Theoretical Computer Science*, 14:317–336, 1981.

[Myers 90]
Brad A. Myers et al. Garnet, comprehensive support for graphical, highly interactive user interfaces. *IEEE Computer*, pages 71–85, Novembre 1990.

[Palanque 95a]
Philippe Palanque et Rémi Bastide. Formal specification and verification of cscw using the interactive cooperative object formalism. *Proceedings of the Human-Computer Interaction (HCI'95)*, pages 213–231, 1995.

[Palanque 95b]
Philippe Palanque et Rémi Bastide. Time modelling in petri nets for the design of interactive system. *GIST workshop on time in interactive systems*, Juillet 1995.

[Paternó 94]
Fabio Paternó et A. Leonardi. A semantinc-based approach for the designe and implementation of interaction objects. *Computer Graphics Forum*, 13(3):195–204, 1994.

[Stasko 89]
John T. Stasko. *TANGO: A Framework and System for Algorithm Animation*. Thèse de Doctorat, Brown University, 1989.

[Van Biljon 88]
W. Van Biljon. Extending petri nets for specifying man-machine dialogues. *International Journal on Man-Machine Studies*, 28:437–455, 1988.

[Wasserman 85]
A. I. Wasserman. Extending state transition diagrams for the specification of human-computer interaction. *IEEE Transactions on Software Engineering*, SE-11:699–713, Août 1985.