Unifying Views of Interactors

David Duke[1], Giorgio Faconti[2], Michael Harrison[1], Fabio Paterno'[2]

(1) Department of Computer Science, University of York, Heslington, York, YO1 5DD, U.K.
(2) CNUCE-CNR,Via S.Maria 36, Pisa, Italy.
{duke, mdh}@minster.york.ac.uk, {paterno, faconti}@cnuce.cnr.it

ABSTRACT
Interactors are components in the description of an interactive system that encapsulate a state, the events that manipulate the state, and the means by which the state is made perceivable to users of the system (the *presentation*). This paper concerns the relationship between the models of interactors that are being developed, at York and Pisa, in the context of Esprit Basic Research Action 7040 (Amodeus-2). The models differ in their expression of the three components of an interactor, and after relating the models to the informal notion of interactor we describe the context in which the view of interaction afforded by each model is appropriate.

1. INTRODUCTION

Much recent work in the field of interactive graphics involves describing the interface of a system in terms of a collection of interaction objects. This approach applies both to the level of implementation [15] (where the term *widget* is sometimes used), and at the more abstract level of system specification [13]. However the notion of interaction object need not be confined to graphics systems; it represents a useful structure for thinking and reasoning about the behaviour of interactive systems in general. As part of Esprit BRA 7040 (Amodeus-2), we are using the concept of interactor, and existing work on state-based processes and agents, to develop models and theories of interactive systems.

Two models are of particular interest. The work at CNUCE is building on experience in describing graphics systems [18] and logical input devices [6] using process-oriented formalisms such as LOTOS. York on the other hand is building on experience in representing and reasoning about properties of interactive systems [10, 11] in terms of their states and displays, and relationships between these.

AVI 94- 6/94 Bari Italy

This report is intended to make explicit the connection between the two models and to describe why it is both appropriate and useful to work with two expressions of the concept of interactor.

Section 2 of this paper concerns the comparison between the models, and we make use of a small example to illustrate both the models in isolation.

Section 3 concerns the connection between the approaches. The two models each emphasise different aspects of interaction, and the formalisms used to express the models afford different approaches to the construction and analysis of specifications. Interactors can also be viewed as architectural models for describing interactive systems. The role of the models in the process of software development is discussed in Section 4; the conclusion outlines the research directions at the two sites.

2. TWO MODELS OF INTERACTION OBJECTS

Our aim in this section is to introduce the two models of interaction objects that are the subject of this paper. Both models describe interactors across abstraction levels, from an external view where the concern is the visible behaviour of an object, to an internal view where the behaviour of an object is explained in terms of some chosen architecture. The task of unifying the two models is properly the concern of Section 3, but to lay an intuitive foundation we will use both models to discuss an exemplar based on material from [18].

2.1 OF MICE AND MENUS

The exemplar used here represents a fragment of the behaviour provided by a graphical user interface, on which objects such as files, disk drivers, and directories are represented by icons. The user can manipulate these using a one-button mouse whose position is represented by a cursor on the screen. We only consider one task, that of selecting and moving icons. An icon is selected by clicking on it, that is, positioning the cursor over the icon and clicking the button, the effect of selecting an icon is to highlight its image. Clicking on a selected icon de-selects it. Selected icons can be moved by choosing the 'move' option from a menu of commands, then picking up one of the icons and dragging it to its new

location. The relative position of selected icons remains the same, thus all selected icons are moved by the same (relative) offset.

2.2 THE CNUCE MODEL

The development of interactors at CNUCE was initially aimed at formalising aspects of the Reference Model for Computer Graphics, in particular, the basic components (interactors) with which interactive graphics programs can be modelled and constructed. Subsequent work such as [18] use the notion of interactor to provide an architectural model for the specification of interactive systems. The approach of [18] is to specify interactors using a process-oriented notation such as LOTOS. At a very abstract level, an interactor is viewed as a 'black box' that mediates between a 'user side' and an 'application side'; it can receive information from either side, process that information, and return it.

Using LOTOS, the external behaviour of such an object is given by

```
process interactor [im,it,is,oc,ot,os] : noexit :=
    oc; interactor [im,it,is,oc,ot,os]
[] ot; os; interactor [im,it,is,oc,ot,os]
[] im; os; interactor [im,it,is,oc,ot,os]
[] it; is; interactor [im,it,is,oc,ot,os]
endproc
```

Informally, an interactor can either:

- receive (and accumulate) output from the application side (*oc*),
- receive an output trigger (*ot*); the interactor then sends output to the user side (*os*),
- receive (and accumulate) input from the user side (*im*) and provide feedback toward the user (*os*),
- receive an input trigger (*it*) that causes the interactor to send the accumulated input to the application side (*is*).

To construct the specification of particular interaction objects, it is convenient to work with a model that makes explicit the relationship between the externally observable events. The model presented in [18] and used within AMODEUS (see for example [17]) is influenced by the computer graphics reference model [12] in its identification of the components that make up an interactor.

The *collection* (see Figure 1) maintains an abstract representation of the information or model manipulated and represented by the interactor. When triggered, it passes this information to the *presentation* which maintains the primitives (for example, basic graphics commands) through which the information is made perceivable to the user side.

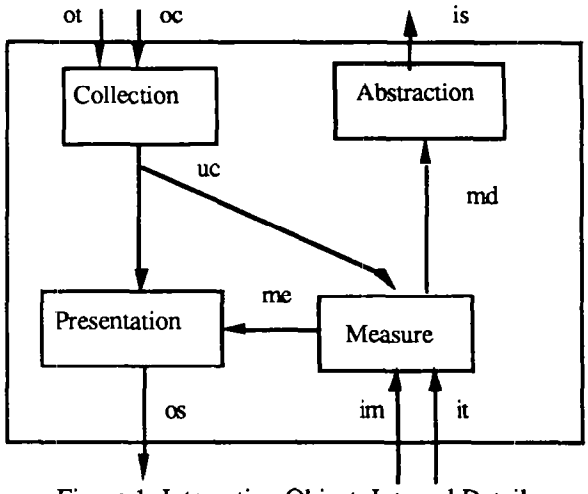

Figure 1: Interaction Object: Internal Detail.

Input from the user is accumulated in the *measure*, which, when triggered, passes this to the *abstraction*, where it is converted into a more abstract representation suitable for distribution to the application. Information received from the user by the measure may be echoed by passing it directly to the presentation. The measure may also receive information from the collection which is used for the processing applied to the input data before passing to the abstraction.

A formal description of the internal view involves specifying each component within the interactor as a separate process. In LOTOS, the actions of processes can be connected to operations on abstract data types, effectively allowing a process to operate on a state. For example, the collection encapsulates and maintains an abstract description of the information displayed by the interactor. This information can be represented by the following data type in Act-One, the algebraic notation used to represent state in LOTOS:

```
type Collection is ListEntity
sorts Collection
opns
empty_col:                      -> Collection
interpret: Collection, Entity   -> Collection
travCol: Collection             -> ListEntity
add:    Collection, Entity      -> Collection
clear:  Collection              -> Collection
eqns ...
```

The equations that define the semantics of the type *Collection* have been elided, but once defined, the data type becomes a parameter to the process specification for the *collection* component:

```
process collection [ot,oc,uc](c:Collection) : noexit :=
 oc?op:entity ; collection[ot,oc,uc](interpret(c, op))
[]
 ot!true; uc!travCol(c) ; collection[ot,oc,uc](c)
 endproc
```

Similarly, the *feedback* component is specified by first defining a data type for pictures,

```
type Picture is
 sorts Picture
 opns
empty_pic:                  -> Picture
mkpict: ListEntity          -> Picture
pick: Picture, ListEntity   -> entity
highlight: Entity           -> entity
```

then linking this to the events in which the process engages.

```
process feedback [uc,me,os] (p:Picture) : noexit :=
uc?p1:entity ; me?id:input ;
 os!highlight(pick(mkpict( p1), id));
 feedback[uc,me,os](mkpict(p1))
[]
 me?id:input ;os!highlight(pick(p, id));
 feedback[uc,me,os](p)
endproc
```

Likewise, we could specify the behaviour and structure of the *control* and *measure* processes, but these can be found in [18]. So far, the specifications have addressed the general architecture and behaviour of an *arbitrary* interactor. To apply the CNUCE model to the specification of a *specific* system, two steps are necessary:

- represent the system as a collection of interactors, and
- describe the data types on which the components of each interactor operate.

Note that it is unnecessary to describe the behaviour of each interactor in the system (that is already defined by the architectural model). It is however necessary to describe how the various interactors that make up the system description should be connected to yield the overall behaviour, and for this the operators defined by the process algebra of LOTOS provide a useful starting point. Recent work at CNUCE [17] has examined operators that are useful for the specific task of building systems from interactors.
The example described informally in Section 2.1 is represented in the CNUCE model by 3 interactors. Our concern here is with the architecture or syntax of the system, shown in Figure 2. The formal semantics would be expressed by defining data types for the subcomponents of each interactor; these details are beyond the needs and scope of this paper, but the interested reader is referred to [18].

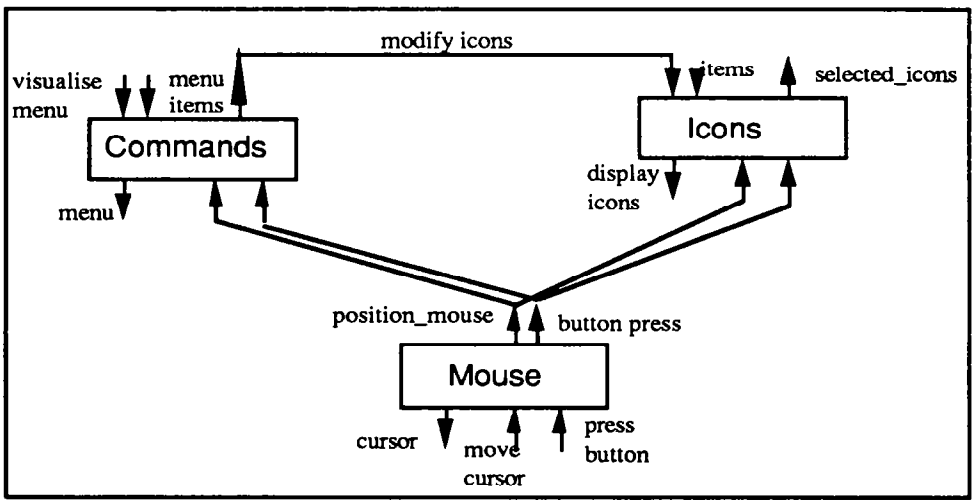

Figure 2: The 'Move Icon' Interactors (CNUCE model).

We can give an informal account of the operation of this system. The user is assumed to provide input to the mouse interactor by moving some physical device and pressing a button; the measure of the mouse represents some position in two dimensional space which is rendered by a cursor which we assume to be perceived by the user. Both of the command and icons interactor use the position of the mouse as input to their measures, and treat the button press event as a trigger to compare the mouse position with that of structures recorded in their collections. The icons collection is a set of icons, and when the position of the mouse at the time of a button press corresponds to the position of an icon the presentation of that icon is suitably modified. In the case of commands, the collection consists of a set of options displayed as a menu, and when the button press occurs with the cursor over the "move" option, the commands abstraction generates an event that causes the icons

interactor to interpret the subsequent motion of the mouse by changing the position of selected icons.

2.3 THE YORK MODEL

The interactor model developed at York has its origins in work involving the use of mathematics to describe principles and properties of human-computer interaction. One outcome of this was the PIE model and its derivatives [4], which talk about interaction by relating sequences of inputs *programs* to *effects* through an *interpretation* function (hence the acronym). In fact at its most abstract, an interactor can be modelled by a structure very similar to that of a RED-PIE, an extension to the PIE model that adds projection functions from the *effect* into representations of the *result* and *display* (see Figure 3 left):

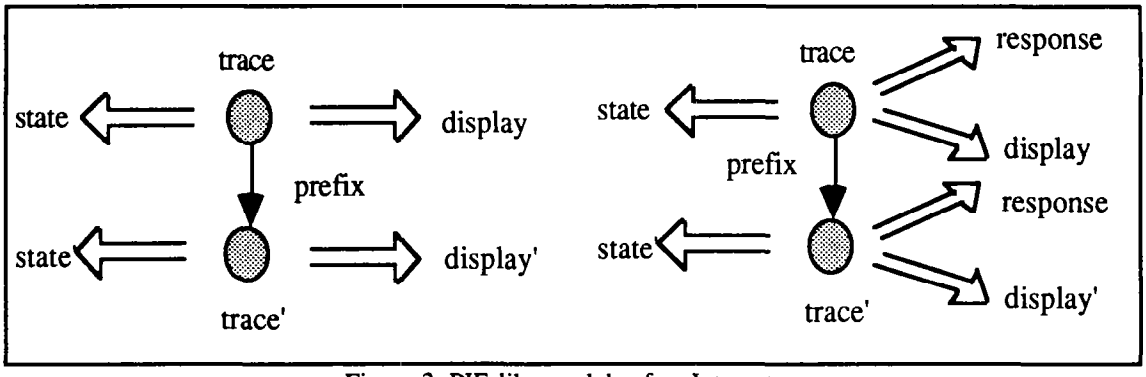

Figure 3: PIE-like models of an Interactor.

Each circle represents some observed behaviour (trace) of a system, expressed as an ordered set of events. Two projections relate each trace to the state and display that the system is in after engaging in the trace, and traces are related by a prefix ordering. Other projections could be introduced for example to represent events generated by an interactor in response to some given input (stimulus), as shown on the right of Figure 3. Indeed, [8] goes further and shows that the behaviour of a system can be described by combining the traces and responses into event structures where some events carry information about the state and display of the system. Formally, an interactor with state and display projections can be described by a 3-tuple:

$$interactor \triangleq (\tau, \_ \overset{R}{\rightarrow} \_, \_ \overset{V}{\rightarrow} \_)$$

where $\tau \in \mathbf{P}\ TRACE$

$$(\overset{R}{\rightarrow}) \in \tau \leftrightarrow S$$

$$(\overset{V}{\rightarrow}) \in \tau \leftrightarrow P$$

$$\forall s, t \in TRACE \bullet s\ prefix\ t \wedge t \in \tau \Rightarrow s \in \tau$$

P TRACE is the set of subsests of TRACE. Here the assumption is that S and P are sets representing respectively the state and display (presentation) space of the interactor. The last predicate (above) requires that the trace set be prefix closed; if some interaction can be observed, then so can any prefix of that interaction.

Although providing a malleable framework for reasoning about interactive systems, the PIE model operates on an abstract view of the external behaviour of an interactor. It does not explicitly model the meaning of individual operations, nor does it by itself address the need in practice to structure the description of systems in terms of well defined autonomous units. This requirement motivated the notion of an agent [1] which combines state-based specification with the use of a CSP-like notation to describe interaction. However agents do not explicitly represent the rendering of the state, that is, which attributes of the state are presented, and the form of this presentation.

The concept of an interactor [9] was introduced to package a PIE-like model that links states, events, and renderings into an agent-like unit that can serve as a building block for realistic specification and development. At an abstract level, an interactor (as pictured in Figure 4)

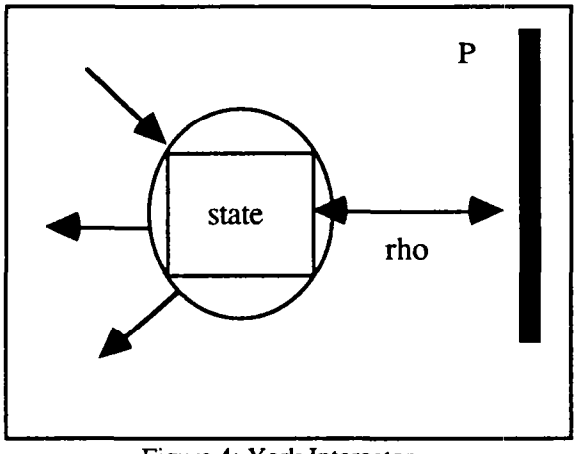

Figure 4: York Interactor.

consists of an internal state which is reflected through a rendering relation (**rho**) onto some perceivable representation (**P**). The interface between an interactor and its environment consists of a set of events. There are

two kinds of events: *stimuli* are caused by agents within the environment and bring about state changes, while *responses* are events generated by the interactor.

Unlike the work at CNUCE which is based on process algebras, the York approach has been to work with (various) logics and the set-theoretic notation Z. To define the concept of an interactor within Z, we begin by introducing a generic model of objects, this will be useful when we come to talk about a refinement of the interactor model.

An object consists of: a state space σ, a set of initial states (ι), an alphabet of events (α), a set of observations or traces (τ), a collection β of state transition relations, and a result relation between traces and states. Here we are assuming that a trace is just a sequence of events. After engaging in trace t, an object may be in any state

The formal description of an interactor extends the object model given above. We introduce a new generic parameter P to represent the space of possible renderings, and define a *view* relation between traces and renderings. For trace t and rendering p, the view relation (v) indicates that p can be perceived after the interactor has engaged in t.

interactor $[S, P]$

object$[S]$

$\_ \overset{V}{\rightarrow} \_ : TRACE \leftrightarrow P$

$\_ p \_ : S \leftrightarrow P$

---

dom *view* $= \tau$

$\forall s : S; p : P \bullet s \rho p \Leftrightarrow \exists t : \tau \bullet t \overset{R}{\rightarrow} s \wedge t \overset{V}{\rightarrow} p$

The rendering relation in Figure 4 is represented in the formal model by the relation ρ between states and renderings. For state s and rendering p, the interpretation of $s \ \rho \ p$ is that after some trace (say t), p is a possible rendering of s. This is a weak notion of state-display conformance; two traces that construct identical states might nevertheless yield distinct views. A stronger conformance can be obtained by requiring that

$$\forall s : S, p : P \bullet s \rho p \Leftrightarrow \forall t : \tau \bullet t \overset{R}{\rightarrow} s \Leftrightarrow t \overset{V}{\rightarrow} p$$

that is, $s \ \rho \ p$ means that any trace that constructs s can result in view p.

A refinement to the York model is obtained by promoting the rendering to the status of a distinct object. That is, rather than view a rendering as a reflection of some state, we can view it *as* the state of a *display object*. Here names that define structures in the state refer to user-perceivable features of the display, rather than the implementable components of a system which the names in a state object refer to.

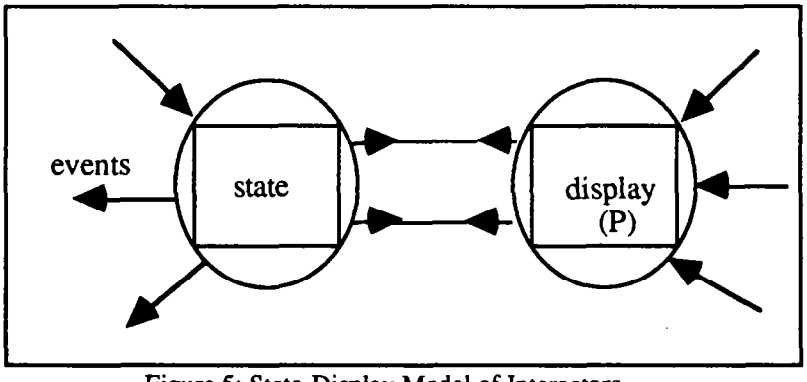

Figure 5: State-Display Model of Interactors.

This refinement step is motivated by the concept of layer conformance which provides an abstract framework for structuring the description of interactive systems in terms of an *arbitrary* number of layers. Layers may include physical systems, data structures, scenes, and, in some systems, a metaphorical layer. An interactor is a structure that encapsulates two specific layers: the state, and the display.

Within the York framework, the 'move icon' exemplar is also described using three interactors (or six objects in the refined presentation). The informal account of the operation of this system is close to that of the CNUCE system, and again we do not attempt to express the semantics of the interactors formally.

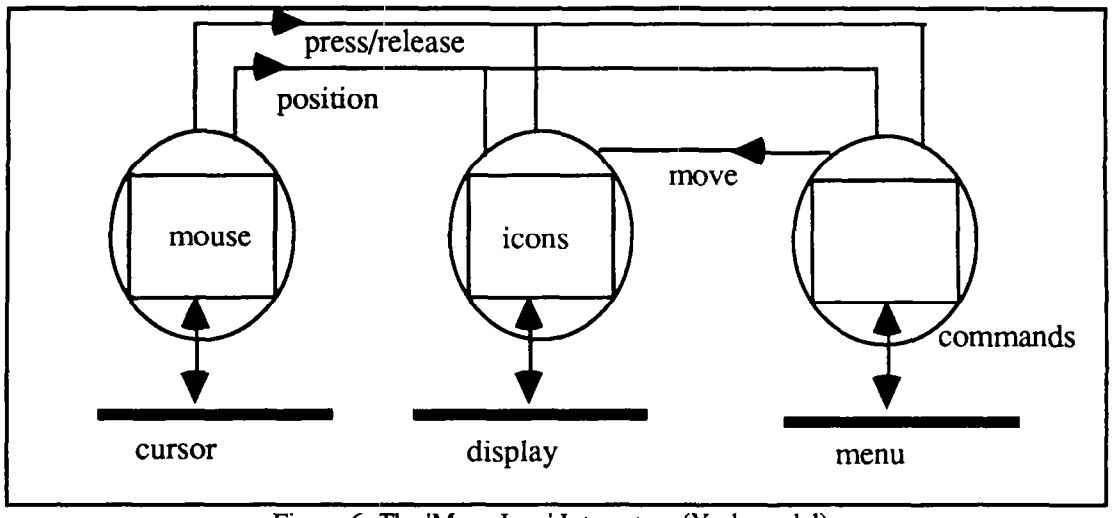

Figure 6: The 'Move Icon' Interactors (York model).

Informally, the components are: a **mouse**, which encapsulates a position and a button status, **icons**, which captures the position and other information related to the icons that can be manipulated, and **menu**, an interactor that presents a list of options and broadcasts an option when it is chosen. The mouse position is rendered by a cursor, while the status of the icons and menu is reflected on a display. Interaction and interference between displays is beyond the scope of this discussion but see [7] for some suggestions.

3. RELATIONSHIP BETWEEN THE MODEL

In the previous section we explained how both the York and CNUCE models are derived from a common understanding of the concept of an interactor. Although based on the one "recipe", each approach has its own flavour brought about by the use of 'ingredients' specific to each site:

- **Notation** Process algebras and logic afford different means of expressing the one concept, and the primitives provided by a (specification) notation influence the way interactors are described and combined.
- **Structure** Each approach has chosen to endue interactors with a particular structure; this difference becomes more apparent once the models are refined.

The purpose of this section is to define the relationship between the two models, in particular, how each captures the structure and features of interaction between system and user. Let us begin by relating the intuitive notion to the two most abstract representations, reproduced below in Figure 7.

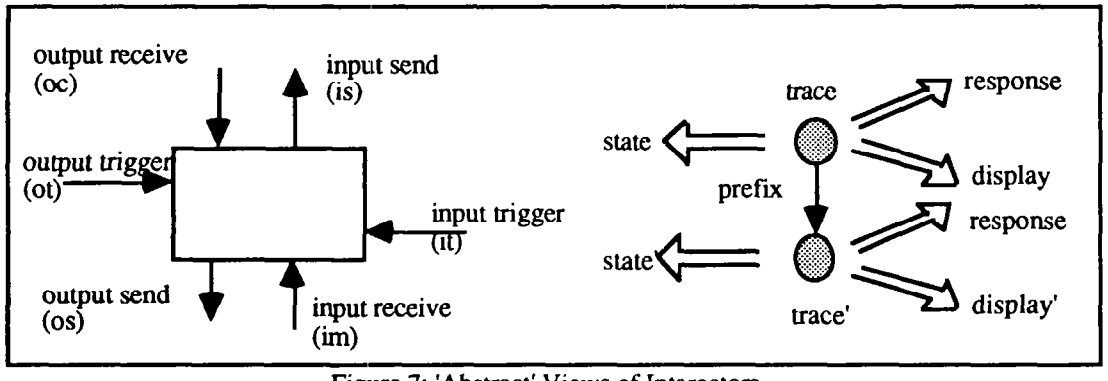

Figure 7: 'Abstract' Views of Interactors.

Both view an interactor as a receptor of stimuli from an environment that can consist of both users and other system components. One difference between the models lies in CNUCE's representation of these stimuli:

- it distinguishes the *source* of events, i.e. events from the user (input receive) and the system side (output receive), and
- it distinguishes between data events (that provide input) and control or triggering events that tell the interactor to do something with the data it has accumulated.

By contrast, at this level the York model treats all events uniformly. For example, consider the representation of the command menu in the two approaches. In the CNUCE model, the *command* interactor receives the cursor position as an input event, but treats the button press as a triggering event which then causes the interactor to interpret the position of the mouse with respect to the options and possibly to send a 'modify icons' event to the *icons* interactor. In the York model, both events are simply defined at the interface of *command*, and the different roles that the events play in defining the semantics of the interactor would only become evident on inspection of the state-transitions associated with each event.

The CNUCE model is symmetric in its treatment of input and output, and with respect to the latter, again distinguishes between events that link an interactor to the user and application sides. Thus the 'menu' event that the *command* interactor uses to display the menu is an example of output send (see Figure 7), while the 'modify icons' event generated by the same interactor corresponds to input send.

While the York model, like CNUCE, describes input in terms of events, its treatment of output distinguishes between the display (which we assume is persistent), and response events, which may communicate information to either the user or some other interactor. Again, the *command* interactor provides a suitable illustration:

- the presentation of the interactor's state is the menu, linked to the state via the rendering relation.
- the selection of an option is an event ('move'), that in the case of the system described in Figure 6 is interpreted by another interactor.

Next, consider the architectural models for interaction objects; for the York approach, we take this to be the one-level model. Figure 8 reproduces the two relevant diagrams:

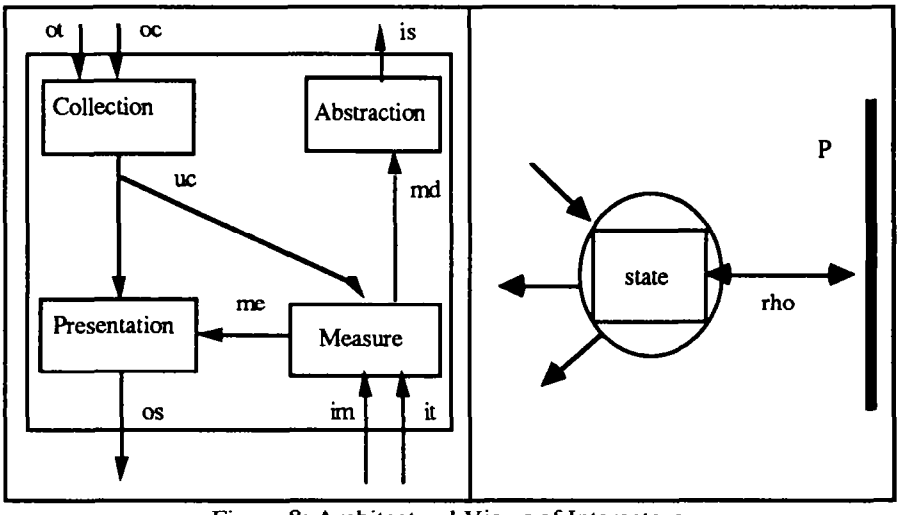

Figure 8: Architectural Views of Interactors.

Superficially, the models appear to have diverged substantially, but this has much to do with the details that are made explicit by the formalisms used to represent the models. In terms of the fundamental concepts underlying the notion of an interactor, we make the following identifications:

- The CNUCE *collection* corresponds to the state of a York interactor; it is an abstract model of the information manipulated and represented by the interactor.
- The CNUCE *presentation* corresponds to the York presentation; differences between the two notions are described below.

The CNUCE *measure*, which maintains a representation of the input received from the user side, has no direct correspondence in the York model. Instead, all events in the York model are interpreted as operations on the state, and thus the state subsumes the notion of a measure. So while the *icons* interactor in the CNUCE model would record the last position of the mouse within its measure, and interpret this position with respect to that of the icons when triggered by the mouse button, in the York model the cursor position is a structure in the state of *icons*, and would be accessed directly in defining the state transition corresponding to the 'press' event.

Similarly, the York model contains no explicit counterpart to the *abstraction*, whose role is to abstract the input structure produced by the measure into a form suitable for distribution to (other) interactors on the application side.

In the York model this kind of abstraction would be encoded in the predicates that relate certain parts of the state (corresponding to accumulated input) to the events that the interactor uses to communicate with its environment. Both approaches model communication

between interactors with a collection of operators; these are not discussed here, but see [7, 17].

Returning now to the presentation-rendering correspondence, there are differences in the two concepts that should be noted. In the CNUCE model, the presentation contains the primitives used to *construct* the representation of the state; these need not correspond to elements of some display, but instead may be commands or structure to be interpreted by some other interactor. This contrasts with the York model, where the presentation component is presumed to represent user-recognisable features of the display. However, this aspect of the CNUCE model can be reconciled with the York approach if we work with the multi-layer model, exemplified by the two-level structure in Figure 5, where the presentation corresponds to the rendering *object* of a state-display pair.

One structure in the York model with no explicit counterpart in the CNUCE model is the rendering relation (**rho**) between state and presentation. This is assumed to relate states to their perceivable representation, or in CNUCE terms, to relate the contents of the collection and the presentation. Instead of using a (static) dependency between system components, the CNUCE model encodes the state-display relationship within the process algebraic description of an interactor. Recall the behaviour of the collection process from Section 2.1:

```
process collection [ot,or,uc] (c:Collection) : noexit :=
or?op:entity; collection[ot,or,uc] (interpret(c, op))
[]
ot!true; uc!travColl(c); collection[ot,or,uc](c)
endproc
```

After receiving an output trigger, the structure in the collection (i.e. the state) is traversed to generate new representations for both the measure and the presentation, and these are passed to the respective processes through the **uc** event. The presentation responds to the collection event by applying some appropriate processing to the input data (such as highlighting a structure). Differences in the encoding of the relationship between state and display within the two models can be traced back to the formalisms used. Z, as a model-oriented formalism, suggests a style of specification where properties and relationships are expressed as predicates over structures. LOTOS and its algebraic sub-language support a more constructive form of expression where the *effect* of events on a system or data type is expressed by equations that define or construct the resulting system.

4. COMPLEMENTARY MODELS FOR SYSTEM DEVELOPMENT

Although the claim of this paper is that the York and CNUCE interactors have a common basis, the two models have complementary roles in the process of software development and in the broader context of developing a principled approach to the design of interactive graphics systems. Within each model, there are also complementary roles for the abstract and concrete representations.

Like the differences discussed in the previous section, the roles are also determined by the architecture adopted by each model, and by the formal notations used to capture the semantics of interaction. In defining roles we identify two *types* of system model:

- **Analytic** models provide a scientific framework for representing and understanding phenomena, for example, various kinds of conformance between states and displays, or emergent features of interfaces.
- An **Engineering** model provides a description of either an existing system or prescribes the structure of a new system. In the latter role an engineering model is sometimes referred to as a 'contract' between client and developer for the construction of some artefact.

Of course the boundary between the two types of model is not exact. Thus, while the abstract representation of the York model (Figure 3) is primarily analytic, it could also be used as an engineering model to express requirements on a system at a high level of abstraction. As an analytic tool, it has been used to give a formal account of the taxonomy of multimodal systems proposed by Coutaz et al, and to express various principles of interaction in a precise language. Both of these activities take advantage of the descriptive power of set theory and predicate logic to define relationships between abstract structures.

The refined York model (see Figure 5) is not in itself an engineering representation, but provides a framework for using model-oriented notations like Z and VDM to specify interactive systems. In addition, it provides a framework for describing the refinement of interactors [8] and thus acts as a bridge between principles and properties in the abstract model and the CNUCE models whose role we consider next.

It has already been noted that process algebras support a constructive approach to specification and thus both levels of the CNUCE model are well positioned to fill the role of engineering notations. Two aspects of these models deserve particular mention in this respect:

- The LOTOS language is supported by a collection of tools that allow specifications to be analysed to detect the possibility of deadlock, for example. [18] describes how the dynamic behaviour of a system of interactors can be explored by unfolding interactions event by event. This kind of investigation may be useful in validating properties of an interface against user expectations.

- The architecture of CNUCE interactors has been informed by work on reference models for computer graphics [12] and systems such as GKS. Refinement from abstract specifications to an implementation in a system may be simplified because the model incorporates the main graphics concepts.

It is important to note that the CNUCE model is supported in the above by its use of events as the sole of representing interaction between user and application (as opposed to York's use of events plus presentation), and by the classification of events with respect to direction and purpose, i.e. data versus triggers. In contrast, the homogenous treatment of events provided by the York models simplifies the statement of properties and should lead to a more tractable framework for reasoning formally about behaviour.

We conclude this discussion by noting that the abstract view of the CNUCE model also plays an analytic role by providing a basis for linking interactors to task descriptions. A similar approach is proposed in the UAN notion of Hartson et al [11]. While this work [17] is focused on using user task decomposition for modelling Interactive Systems described by the uniform treatment of interaction provided by the CNUCE model.

CONCLUSIONS

For software development to take on a formal approach to user-oriented requirements there needs to be a scientific framework in which those requirements can be stated precisely, and there needs to be a framework for systematically expressing these requirements as part of the development process. We claim that the notion of an interactor provides a suitable framework for expressing both analytical and engineering models of interaction. Within the AMODEUS project the development of the interactor concept at York and CNUCE has lead to various expressions of the underlying concept that are each appropriate to particular tasks in development. Figure 9 summarises these roles (for comparison we have also indicated the position of PAC [2]).

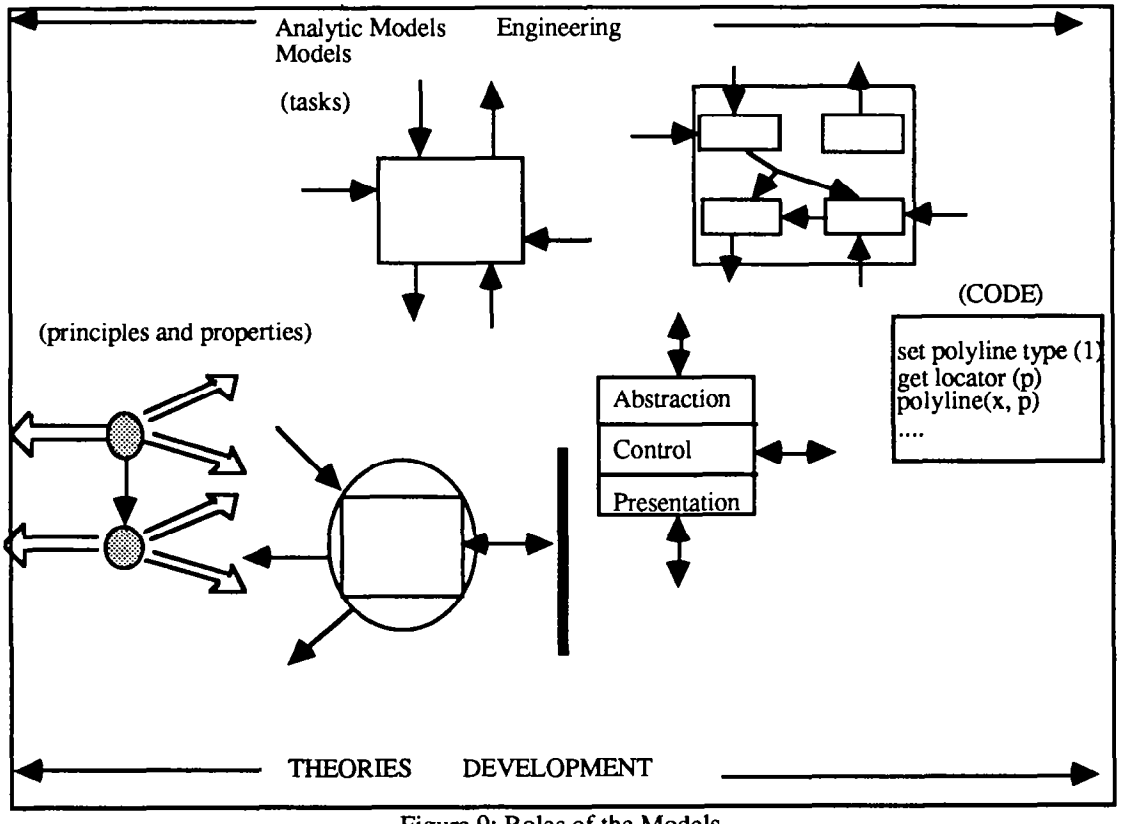

Figure 9: Roles of the Models.

Work at CNUCE is currently focused on developing the link between interactors and task models, on understanding how interactors can be combined to capture the behaviour of complete systems and to apply tool-supported verification techniques for verifying usability properties. At York the concern is with the role of interactors in a rigorous development process, expression of properties related to ease-of-use, and refinement of interactors towards implementations.

REFERENCES

[1] G. Abowd. Formal Aspects of Human-Computer Interaction. PhD thesis, Oxford University Computing Laboratory: Programming Research Group, 1991. Available as Technical Monograph [PRG-97].

[2] J. Coutaz. PAC, an object-oriented model for dialog design. In H. Bullinger and B. Shackel, editors, Human-Computer Interaction - Interact'87, pages 431--436. North-Holland, 1987.

[3] J. Coutaz, L. Nigay, and D. Salber. Conceptual software architecture models for interactive systems. Technical Report WP11, ESPRIT BRA 7040 Amodeus-2, March 1993.

[4] A. Dix. Formal Methods for Interactive Systems. Academic Press, 1991.

[5] D. Duce, M. Gomes, F. Hopgood, and J.Lee, editors. User Interface Management and Design. Eurographics Seminars, Springer-Verlag, 1991.

[6] D. Duce, P. ten Hagen, and R. van Liere. An approach to hierarchical input devices. Computer Graphics Forum, 9(1):15--26, 1990.

[7] D.J. Duke and M.D. Harrison. Abstract Interaction Objects. Computer Graphics Forum. NCC Blackwell, Vol.12. N.3. pp.25-36.

[8] D.J. Duke and M.D. Harrison. Rigorous development of interactive systems. Technical Report WP16, ESPRIT BRA 7040 Amodeus-2, May 1993.

[9] M.D. Harrison. A model for the option space of interactive systems. In Engineering for Human-Computer Interaction: Proc IFIP WG2.7 Conf. Elsevier, 1992.

[10] M.D. Harrison and A. Dix. A state model of direct manipulation. In M.D. Harrison and H.W. Thimbleby, editors, Formal Methods in Human Computer Interaction, pages 129--151. Cambridge University Press, 1990.

[11] H.R. Hartson and P.D. Gray. Temporal aspects of tasks in the user action notation. Human-Computer Interaction, 7:1--45, 1992.

[12] Geneva ISO Central Secretariat. Information processing systems, computer graphics, computer graphics reference model. ISO/IEC DIS 11 072, 1991.

[13] R. Jacob. A specification language for direct-maniplation user interfaces. ACM Transactions on Graphics, 5(4):283--317, 1986.

[14] G. Krasner and S. Pope. A cookbook for using the model-view-controller interface paradigm in Smalltalk-80. Journal of Object-Oriented Programming, August-September 1988.

[15] B. Myers. A new model for handling input. ACM Transactions on Information Systems, 8(3):289--320, July 1990.

[16] L. Nigay and J. Coutaz. Building user interfaces: Organizing software agents. In Proc. ESPRIT'91 Conference, 1991.

[17] F. Paterno'. A methodology to design interactive systems based on interactors. Technical Report WP7, ESPRIT BRA 7040 Amodeus-2, February 1993.

[18] F. Paterno' and G. Faconti. On the use of LOTOS to describe graphical interaction. In A. Monk, D. Diaper, and M. Harrison, editors, People and Computers VII: Proc. of the HCI'92 Conference, Conference Series, pages 155--173. British Computer Society, 1992.

[19] F.Paterno'. A Theory of User-Interaction Objects. CNUCE Internal Report. December 1993.

[20] G. Pfaff, editor. User Interface Management Systems. Eurographics seminars, Springer-Verlag, 1985.