Chapter 2.2

Analysing Display Oriented Interaction by means of System Models

M. D. Harrison, G. D. Abowd* and A. J. Dix

Human Computer Interaction Group, Department of Computer Science, University of York, Heslington, York, YO1 5DD, UK

1. Introduction

It is clear that software engineers should design and implement interactive systems with their users in mind. At the same time system evaluators should have a precise understanding of the system if they are to understand the consequence of their evaluation. In this chapter we consider instances in the use or design of graphics based systems so that we might understand how formal specification techniques can be used to assess the design implications of usability issues more effectively. A commitment to formal specification is presumed here in the sense that we assume, without discussion, that the specification of a system should be expressed as a collection of mathematical objects. This view concurs with an increasing number within the software engineering (see, for example Nichols, 93). The advantages of such an approach are precision and clarity of expression, mathematical tractability and the provision of an established set of rules for refining a specification to a computer program. Given formal specification as our baseline, it is natural that we assume that system requirements concerned with usability should be expressible as properties of such specifications. It is also to be assumed that we would wish to be able to demonstrate that such properties hold true.

* Current address: School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213.

We recognise that there are substantial problems of scale associated with large formal specifications. However the potential advantage of a formal approach are such that we believe it is important to explore the development of formal specifications for a wide domain of application and variety of purpose. It is worth noting that the use of formal specification techniques is commonplace within other engineering disciplines.

Formal specifications of realistically sized systems can be large and as a consequence the proof that the specification satisfies given properties is complicated. Complexity can be reduced by including in the design process more abstract descriptions which emphasise aspects of behaviour such as performance, interactivity and security. Models may be specified as a basis for expressing and capturing these properties in the design of the system. Here we are concerned with the specification of the external behaviour of interactive systems; hence, our models are referred to as *interaction models*. Our aim in this chapter is to show how such descriptions may be used to formulate *user recognisable* structures to provide criteria for designing and evaluating interactive systems. In the next chapter the related problem of analysing user requirements and applying them to system specifications is discussed. The user recognisable structures here are less concerned with functionality and more concerned with the nature of the interactive behaviour of the system. We contend that formal models which are specific to the issues of interaction can clarify those parts of the system specification that affect its interactive properties. Thus we offer an approach to the formalisation and integration of user requirements into system specification.
In this chapter, we shall demonstrate our approach through two *scenarios* that have been chosen because they demonstrate different aspects of the requirements on interaction models. The scenarios arise as result of the research of the ESPRIT Basic Research Action project 3066 (AMODEUS), and were initially posed as problems for different modelling techniques in HCI research (Hammond & Myers, 1990). The scenarios relate to graphical drawing packages implemented on personal computers, specifically SuperPaint for the Macintosh and MicrografX Graph Plus for the IBM PC. Fragments of user behaviour are employed to demonstrate problems in the use of those systems. In both cases, the features of the scenario can be abstracted away from the particular implementation, as they both represent common problems that can arise in a variety of graphical drawing packages. We will provide more abstract descriptions of the

scenarios based on these package dependent descriptions to avoid confusion that might arise as a result of unfamiliarity with the packages.
The rest of this chapter is divided into two main sections, one for each scenario. To discuss the first scenario we will derive a description of the relationship between the system state and the display that is factored into task relevant features of both, called *attributes*. This leads to a formal description of *state display conformance*, a principle of the interactive system which guides our analysis. The second scenario involves the relationship between the *history* of interaction and *predictability*. We will suggest ways in which we might analyse how the user predicts the outcome of future interaction with the system.
Though the discussion of a scenario concentrates on a particular interactive property - be it state display conformance or predictability - the impact of these properties can be seen in each. For this reason, it is insufficient to discuss a scenario in terms of only the one property. For purposes of presenting the models, we must forego a desire to be complete, though we will occasionally point out potential overlaps between the scenarios.

2. Relating attribute algebras in a layered graphics package

2.1. Scenario description

Many drawing programs support *layers* for the construction and manipulation of pictures. These layers should not be confused with the levels used by many systems to simulate a 3-dimensional structure. A layer in our sense is essentially an independent canvas upon which pictures can be constructed. The two layer system of concern in this scenario contains an *object* layer, in which text, boxes, circles, etc. can be created and edited, and a *pixel* layer, in which freehand pictures are constructed and manipulated at the lowest level of screen detail (the pixel). The design of the system has a number of interesting features but for our purposes we note the following:

- There are commands that relate distinctively to the two layers. For example, within the pixel layer *paint* and *rub out* commands are used, and their function only makes sense in the pixel domain. Within the object layer, the *select* and *group* commands are used, and their function only makes sense in the object domain.

- There are a large number of commands that the two systems appear to have in common, i.e., they create visual images that appear to be identical and the commands are invoked by identical icons.
- The display representations created at the two levels are often ambiguous and therefore it is not clear what operations are appropriate at which level.

We will explore the last feature with the aid of an example from SuperPaint. A circular image can be drawn within either layer. Figure 1 shows a picture of two circles which have been drawn in separate layers using the circle drawing facility in each layer. It is impossible to identify, visually, which circle belongs to which layer.

The icon in the upper left-hand corner is an indication of the active layer, i.e., the one whose construction and manipulation commands are currently indicated in the palette on the left side of the screen. If the user wants to move the circle on the left, she must make sure that she is in the appropriate layer to manipulate that circle. The circle on the left provides no clue about the layer in which it was constructed and as a result the user must either remember how it was constructed or guess. In terms of the predictability, which will be discussed in more depth in the next section, because there are ambiguities in the display, there is not enough information for the user to unambiguously determine how her future interactions will affect the system.

Although it might be argued that in its current context this is a trivial problem, it is not difficult to imagine situations in which ambiguities of this kind might lead to uncertainties that could lead to error in critical situations.

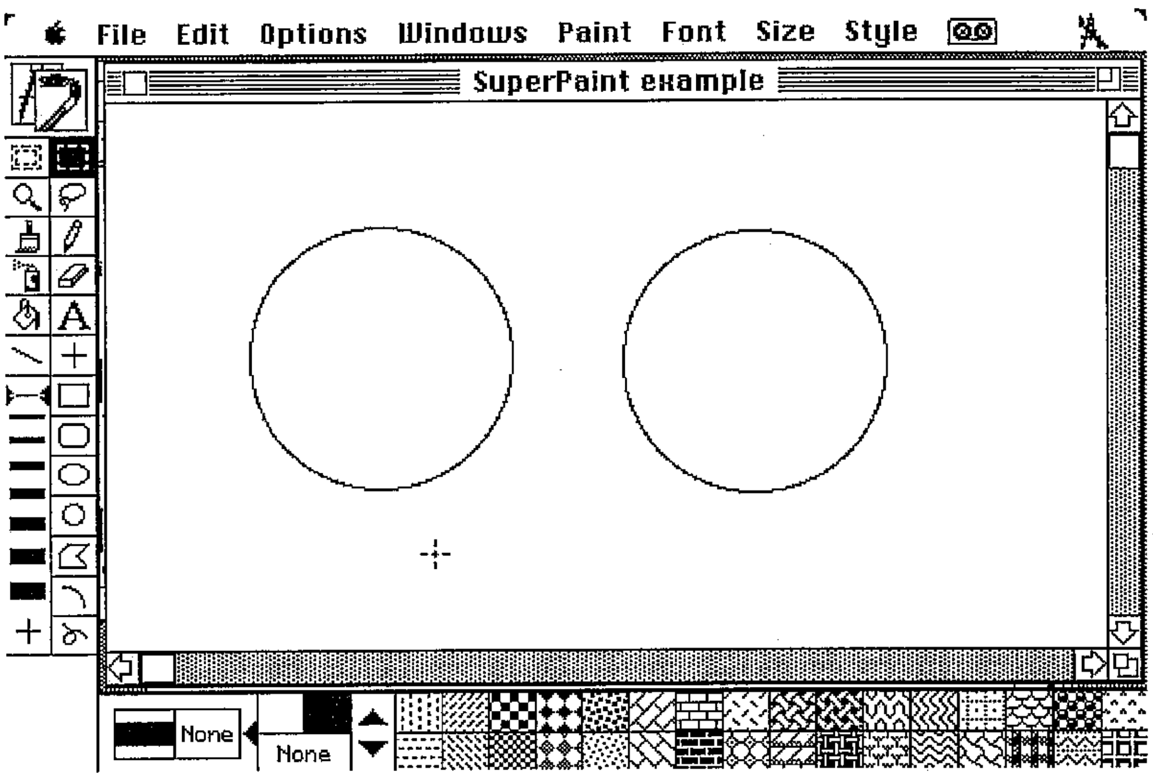
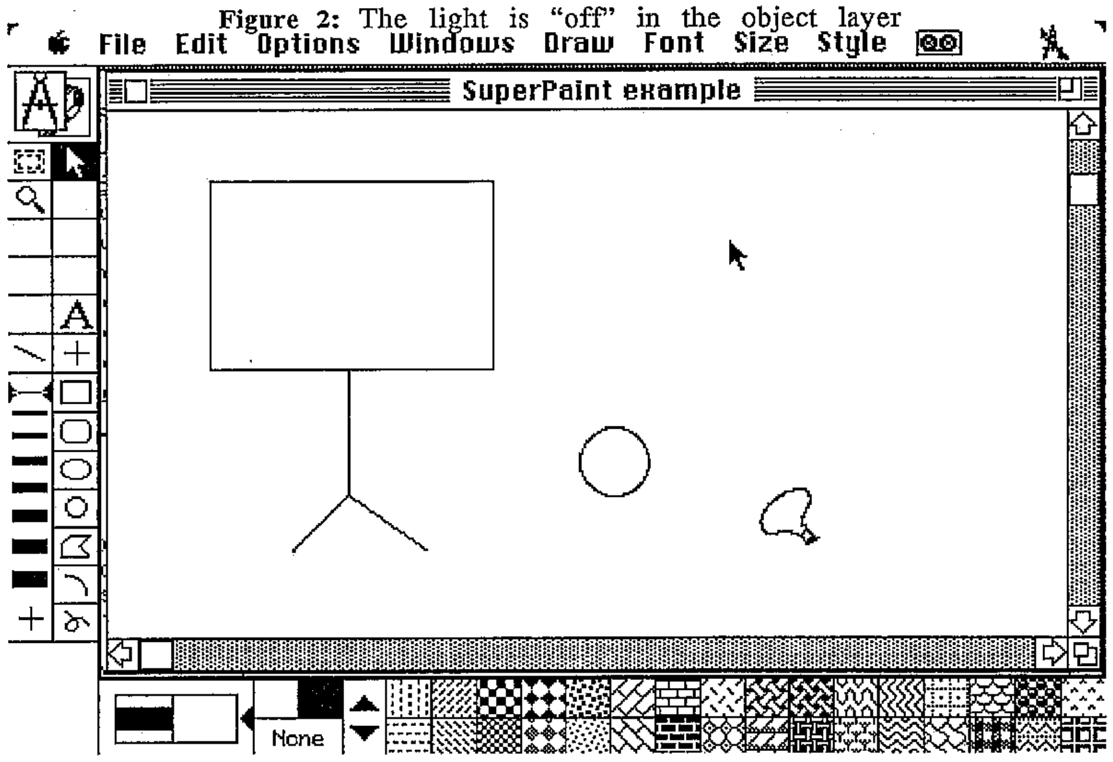


Figure 1: To which layer does each circle belong?

A more complicated situation in which confusion arising from layer ownership affects interaction is as follows. Figures 2 and 3 show how switching between layers can produce interesting variations. The difference between the two canvases displayed in Figures 2 and 3 is a result of the difference between system modes associated with object construction and picture construction respectively. What is contained in the underlying state of the picture is the same in both cases, but the display associated with the object and picture modes are different because the image created in each layer if it overlaps something constructed in the other mode occludes it. In the object mode the "screen" and "light bulb" are visible as entities containing a "white" canvas. In the paint mode we can see that the wide backgrounds of the object layer have hidden circles and filaments created in the picture mode. That these different objects were constructed in the different modes is not clear from the canvases, nor is it easy to construct additional components of the state if the information that the additional objects relate to were constructed in the other mode and are now hidden from view.

Figure 2: The light is "off" in the object layer



The picture created in figure 2 is embellished by adding the information to depict the effect of the light being switched on using the paint layer.

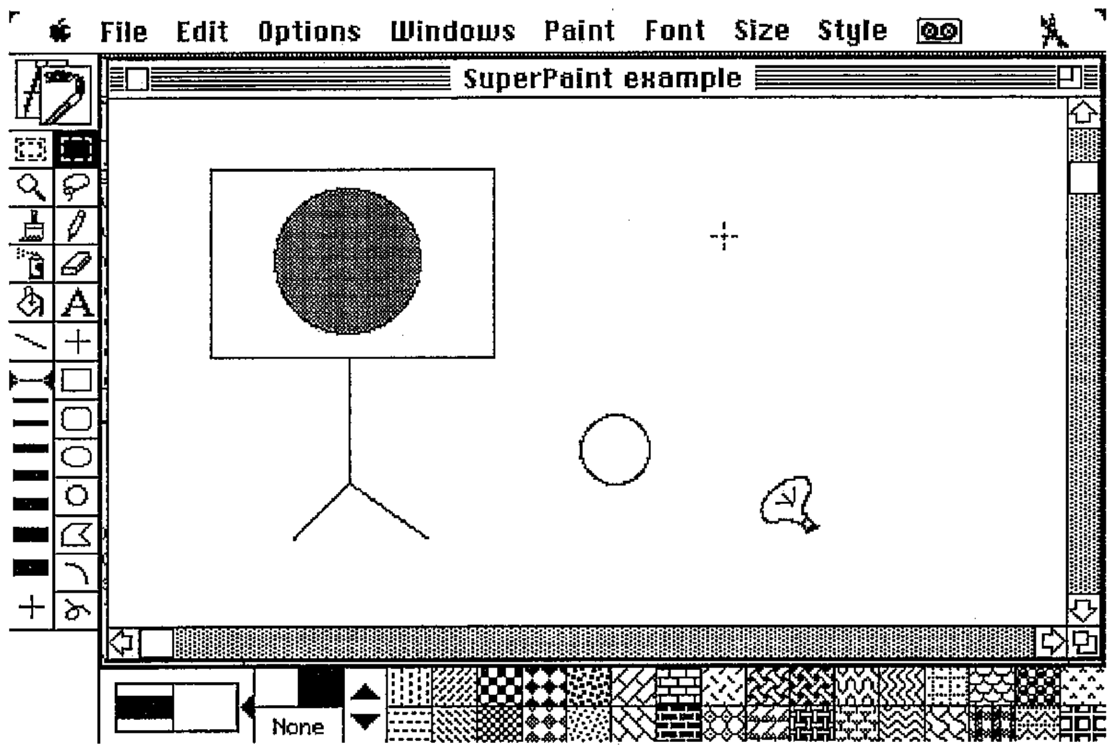


Figure 3: The light is "on" in the paint layer

2.2. Formal analysis

At the most general level, the state of the system, represented by the set S, and the state of the display, represented by the set D, are related by a *view* relation.

$$view: S \leftrightarrow D$$

Operations are defined as relations on a state set. System operations are relations on S and display operations are relations on D. Hence the operations that are associated with the state and the display are taken from the power set, P(), of state and display relations.

$$S\_Ops \equiv \mathrm{P}\,(S \leftrightarrow S)$$
$$D\_Ops \equiv \mathrm{P}\,(D \leftrightarrow D)$$

The two state sets are said to conform if for each operation on one state set, there is a corresponding operation on the other state set. When an operation on the system state, $c \in S\_Ops$, corresponds in such a way with an operation on the display state, $o \in D\_Ops$, we have the relation

$conforms: S\_Ops \leftrightarrow D\_Ops$

$(c,o) \in conforms$ **iff**

$\forall\, d \in \mathrm{dom}(o);\ s \in \mathrm{dom}(c) \mid (s,d) \in view \bullet$

$\quad \exists\, d' \in D;\ s' \in S \mid (d,d') \in o$ **and** $(s,s') \in c \bullet (s',d') \in view.$

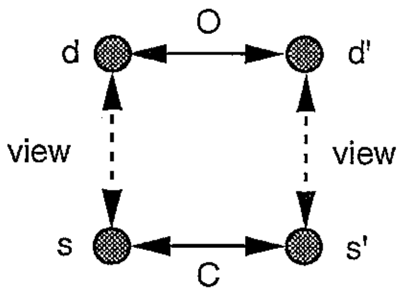


Figure 4: The conformance diagram

The full property of state display conformance (Harrison & Dix, 1990) requires that every operation on the system state has a unique operation on the display state that corresponds with it in the way defined above, hence in their case *conformance* is a more constraining relation.

This relation between state and display does not take into account how the user's task affects state display conformance. So, assume the set T represents the tasks of interest for a given domain of application. We now proceed with a refinement of the definition of state display conformance that factors the system and display

states into more manageable and, to the user or designer, semantically meaningful components, called *attributes*.
In practice, computer users often have a different picture of the system from that intended by the designer. Users are concerned with performing tasks. As is described in Chapter 3.1, these can be broken down into a set of goal tasks (which the user wishes to accomplish) and enabling tasks (which allow goal tasks to be performed). When performing a task the user recognises that components of the state are relevant to the *goal* of the task. For a goal task, this will be something that the user is interested in accomplishing. For an enabling task, it will be something that brings about a state in which a goal task can be performed - and so they might not necessarily be attending to the appropriate components. This recognition is reinforced by components of the display. As the components of the state are modified in progress towards the task goal the display is used to emphasise the fact that progress is being made and to help the user formulate the next command whether it carries out a further enabling task or is actually part of their goal task. Progress toward the satisfaction of a goal, therefore, is enhanced by a close correspondence between system and display as described above.
Assume we have a set of all possible attribute names, denoted by A, and a set of all possible attribute values, denoted by V. A state is simply defined as any finite partial mapping from attribute names to values. For a given interactive system, we stipulate that all states are defined over the same set of attributes. In other words, if the set of system states is given as a set of finite partial attribute-value mappings,

$$S \equiv \mathrm{P}(A \rightarrow V),$$

then the domain of every state's attribute-value mapping is the same, or

$$\forall\, s,s' \in S \bullet \mathrm{dom}(s) = \mathrm{dom}(s').$$

We refer to this domain as the attributes of the system state set S, and we write this as $attr(S)$. Similarly, we can refer to the attributes of the display state set D, and for this we write $attr(D)$.
In recognising display attributes of the system, the user engages in two processes: identification and interpretation. In the case of identification, the display attribute is considered in a way that is similar to the notion of *display template* (Harrison, Roast & Wright, 1989), a notion that is concerned with the component of the display that is relevant to the task. The interpretation captures the significance and meaning of the attribute. We assume therefore that a display attribute has two components: a positional component, and an interpretative component. Hence if the attribute happened to be the date we would want to know something about the date and how it fits on the display (for

example, the top right hand corner) but we would also need to know more than that it is a grouping of pixels. The display attribute illuminates the state attribute, that is a display attribute may reveal information about a particular attribute of the state. Hence the state attribute for the date will extract the field within the state that contains the internal representation of the date.
We note that another attractive feature of the attribute model is that it resembles notions of semantic features used by user modellers, for example, in the work of Young & Whittington (1990) and Payne & Green (1986).
We can refine our model so that it uses state and display attributes that are relevant to the tasks in T to determine the view relationship. We are interested in how tasks can be used to constrain the relationship between system and display attributes. This can be represented by the mapping *taskview,* which specifies a set of task views that relate individual state attributes to display attributes.

$$taskview: T \rightarrow (attr(S) \leftrightarrow attr(D))$$

Hence, for any task t, the connection between relevant task and display attributes is established. We can use this mapping of attributes to represent a claim about how a typical user understands the effects of a set of operations and thus it can be seen as a component of a designer's view of the user's model of the system.

A given operation transforms the state; a designer specifies an operation in terms of the changes that it causes to the attributes. These changes to attributes are also expressed in terms of the value of certain attributes of the state. We refer to the ***signature*** of an operation as those attributes which can be changed by the operation and those attributes whose value before the operation determine the effect of the operation. We can choose the signatures for any operation on the system state, representing this by the function *S_sig.*

$$S\_sig : S\_Ops \rightarrow \mathbb{P}\,(attr(S))$$

Hence the state signature associates a set of state operations with the attributes of the state that they affect. The function *D_sig*, represents the display attributes that are affected by operations on the display.

$$D\_sig : D\_Ops \rightarrow \mathbb{P}\,(attr(D))$$

If we were also to relate attributes and signatures to task, and to user perception, then we could define the signatures that the user might recognise which might be different from the actual signatures. We can use this information to provide a component of a claimed user model if we contend that the user understands the performance of operations on the display state in terms of signatures. Now, if an operation on the system state is to correspond with an operation on the display state, then it will only be recognised by the user if the signature of the state operation is related by the current task via *taskview* to the signature of the display operation. Ultimately, this leads to the user's formulation of the state display correspondence in terms of the task, which we represent as the function

$$user\_conforms : T \rightarrow (S\_Ops \leftrightarrow D\_Ops) ,$$

We may further define this relationship in terms of the signatures of operations on system and display state. The interactive system has the property of state display conformance for a given task, *t*, if *user_conforms(t)* is not defined for all state operations, but no pair of state operations maps to the same display operation (in technical terms a partial injective function). We only require a partial function because some operations on the system state may not be relevant to the execution of some tasks.

In the scenario described above, we can split the system up into two separate sub-states, one for each layer. In isolation, the interactive system may well satisfy the constraint that for a given task, *t*, we have that *user_conforms(t)* is a partial injective function. This is so because in isolation, the attribute of an entity that identifies it as belonging "object" or "pixel" layer is not contained in the signature of the operations on the system state, so it need not be displayed. When the two systems are combined, this attribute becomes part of the signature of many operations on the state, but it is still not displayed. Therefore, the user does not benefit from a strong state display conformance. Using these sorts of techniques the designer develops a claimed model of how the user understands the system.

2.3. Conclusions

One critical issue which we have highlighted in the discussion of this scenario is that the user relies on a strong correspondence between the operations performed on a display and those performed on the underlying system state. This notion has been previously formalised and presented as a state display conformance. The original formulation of state display conformance (Harrison & Dix, 1990) gave us a start in pinpointing the problems that arise in interacting with the overlapping graphical layers. We further refined this definition in terms of attributes which highlight features of a state that can aid the user in understanding how a system work. The attributes are also used by designers in specifying both display and system states,

and the user-centred design principle represented by the function *user_conforms* provides a constraint on the design that can lead to more effective interaction.
The determination of these attributes is one of the key areas where the formal and informal elements of the design process meet. Determining what parts of the state are important to the user will involve some elements of task analysis. Similarly, determining the appropriate display attribute requires some notion of *salience* that requires either the input of an experienced designer or experiment. This can also be related to other work reported in this book. If the attributes of the system can be formally specified then this makes it possible to identify accurately the values of the "machine" parameters in the usability principles that are presented in Chapter 3.2. The prescriptions of these principles allow design expertise and empirical human factors data to be codified and made available at an appropriate point in the design process.

The scenario that has just been discussed is essentially static in the sense that it relates to the user's current understanding of the state of the system as a result of what she sees. The next example deals with properties of the system that enable the user to predict the effect of an operation on the system.

3. Dynamic properties of the interface: determinism

3.1. Scenario description

We describe an object-oriented drawing package with a graphical interface. The objects in a picture have three relevant hierarchies, a structural hierarchy, which is dictated by grouping achieved in the process of constructing the picture, a visual hierarchy, which is dictated by current positioning on the canvas, and a temporal hierarchy dictated by the order in which objects are created. The following sequence of actions in relation to the use of a drawing package demonstrates the relationship between the three hierarchies. It should be noted that in this example grouping is only carried out in the last step of the sequence.

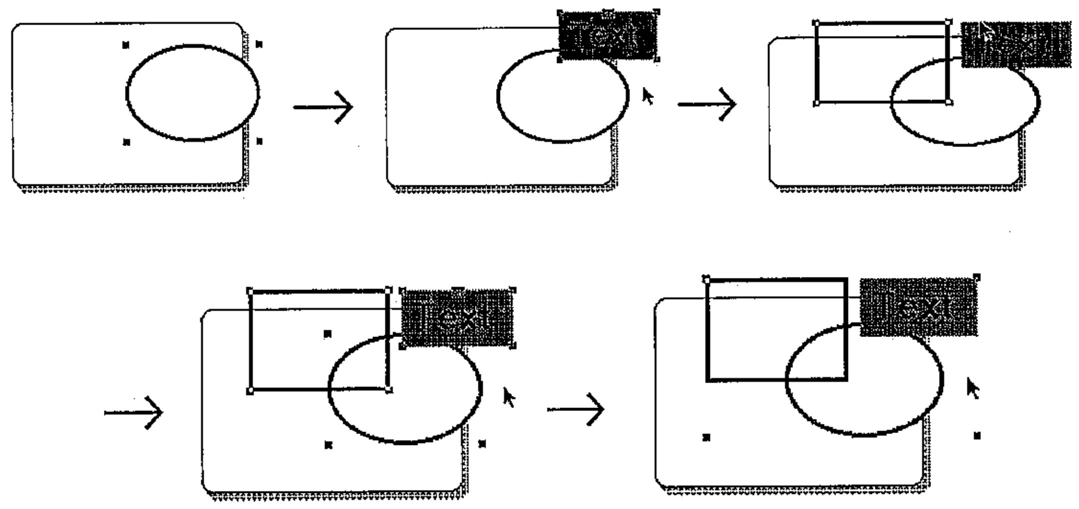


Figure 5: constructing the diagram, the temporal sequence.

In this series of pictures the user has created an oval, then created a Text shaded square, then created a rectangle, then selected all three objects and finally grouped them. The operation of selecting an object is complicated by this distinction between structural, visual and temporal hierarchies. The user manual for this system informs the user that a mouse click enables selection. As it happens, the selection cycles amongst the objects that contain the mouse pointer. We may presume, therefore, that the operation of clicking will select an object containing the mouse pointer; each subsequent click will select another object containing the mouse pointer. After each possible object has been selected, the first object will be selected again, and so on. For example, if the mouse were positioned as below and no objects were currently selected, the selection cycle might go square, circle, grouped square/circle/text, shadowed box, and back to the beginning.

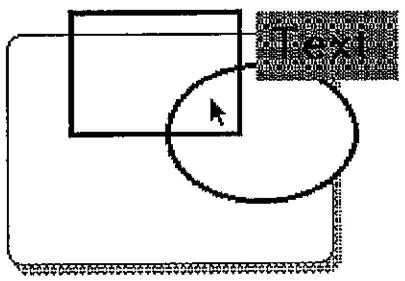


Figure 6: If the mouse is positioned as above what will be selected?

The goal in analysing this scenario is to explain why even though the system's algorithm for selection is deterministic, it is often perceived by the user as non deterministic. The user's response to such perceived non determinism is to adopt the approach "I will just keep on clicking the mouse until the object I want to select is finally selected." This "click until you drop" strategy assumes that the user can always tell when an object is selected. There are likely to be design possibilities in which the user cannot recognise which subset of the diagram has been selected. In this case the lack of visual feedback will mean that this experimental approach is inappropriate.

3.2. Formal analysis

An interactive system, as we have said, is used to achieve a set of task goals. We might informally consider a predictable system to be one which supports these tasks by providing enough information (in sound or vision for example) to indicate what the effect of a new action will be. It is important to recognise that there will often be a difference between the actual behaviour of the system and the user's perception of the behaviour of the system. Predictability of a system can operate at a number of levels: at one level the system may be unpredictable in that it fails to be deterministic as a system (for example executing a command might have an arbitrary outcome); it may be unpredictable because it is impossible to tell from the outward manifestation of the system what effect a particular action will have; it may be entirely predictable in fact but be apparently unpredictable to the user because of the way that relevant information is presented by the system. Hence in the last two cases, the possibility of non determinism is a matter of user interpretation.

System non determinism

A computer system may be unpredictable because an input sequence does not have a unique effect on the state of the system. This notion of system non determinism may be expressed as follows:

> *interpret* maps the input sequence to a set of possible state transformations. The choice between the alternatives is arbitrary.

It is hard to imagine a design that would be non deterministic in this sense unless the hardware upon which it was based was very unreliable. However, if we consider certain kinds of dynamic system in which autonomous activity may complicate the task for the operator, something like system non determinism is true of the system from the perspective of the user. In practice we are more interested in other more subtle forms of perceived unpredictability that do not correspond to non determinism of the system. Here we are concerned for example about whether the system provides enough information on the display to allow the user to predict what effect a command will have. These notions of perceivable predictability capture the system's ability to express enough information about its underlying state for the user to make use of the system's determinism.

Observational equivalence, indistinguishability and predictability

A system is observationally predictable if current display and future input fully determine the resulting behaviour. Imagine that two identical versions of the same system are represented by two keyboard/display pairs. At each keyboard a different input sequence is issued by the user, sequence *p* to system 1 and sequence *q* to system 2, after which the displays, or current outputs, are identical. In this case, we say that *p* and *q* are *observationally equivalent.* The same sequences *p* and *q* are *observationally indistinguishable* if any input sequence *r* subsequently issued to both systems results in the same final display, that is, any further action will betray no difference between the two versions of the system.
A system is *observationally predictable* if all observationally equivalent sequences are observationally indistinguishable. Hence, for a given display, a user can in principle predict the future behaviour of the system because the state of the system is somehow characterised by what can be perceived of the current display. However it should be noted that, apart from certain types of system, where the full state is represented in the display, the observational predictability property is unrealistic, since the underlying state may be far too complex to display on one screen. In fact we may generalise this notion of predictability by defining it in terms of sets of display attributes. A system will be predictable to the user if all the information is there on the display and the user can appropriately interpret it as a means of deciding how a command will function. Hence the drawing package described earlier could be predictable with respect to the drawing and selection operations if the effect of these operations was entirely governed by the content of the display.

User non determinism

At the third level it is important to determine whether the user actually benefits from the predictability of the system. The user may benefit in two ways:

- Does the user's effective interaction with the system rely upon its predictability?

- Is the information that is provided to inform the user sufficient (given cognitive constraints) to predict the future effect of the system?

Just because a user can predict what output will arise from a given input, does that improve interaction with the system? There is a complementary notion to predictability, which we might call assessment, that deals with reasoning from perceived outputs back to inputs. When a user interacts with a system, achievable goals are continually being assessed within the task domain. In order to accomplish the goals it is necessary both to be able to predict how subsequent inputs will affect the goal, and how a goal was (or was not) affected by prior inputs. In addition, experimental evidence may show that predictability provides no measurable increase in the usability of a system. Here we assume there are situations in which predictability does enhance usability. Conversely the system may appear to be unpredictable because features of the display (attributes) required by the user are not appropriately identified and interpreted by the user at the time that they are required.

In Chapter 2.3, Blandford, Harrison and Barnard present a framework in which these properties of the interaction that cannot be expressed in terms of the system may be represented. Properties are defined in terms of the relationship between collections of information exchanges.

Predictability is defined in terms of:

- a sequence of inputs that is interpreted as a state transformation;
- the initial states;
- a display of the state after the transformation.

The interpretation is denoted by a function $interpret : P \rightarrow S\_Ops$, where we represent non empty sequences of input as coming from the set of programs P and the state transformations are denoted by the set S_Ops defined as above. At a finer level of detail, the input is formed from a set, I, of all possible user inputs - for example, keystrokes, mouse movements, mouse clicks, etc.- so that we can represent P as the set of non empty sequences of elements in I. In Chapter 2.3 we use the events, message exchanges between agents, rather than input sequences. The display is again represented by the *view* relation, which for simplicity we assume is a function in the remainder of this account. For simplicity's sake again, we will assume a unique initial state, s_0. We can express observational predictability described above in terms of this model.

A system is *observationally predictable* if, for any input sequences p and q satisfying

$$view(interpret(p)(s_0)) = view(interpret(q)(s_0)),$$

we have for any subsequent experimentation using input sequence r,

$$view(interpret(pr)(s_0)) = view(interpret(qr)(s_0)).$$

An important class of systems that are not observationally predictable have the property that the history of the interaction is required in order to predict what the next command will do. We can capture this notion of history based predictability by defining the input sequence as $p = a_1a_2...a_n$ and a relevant sequence of states as

$$s_i = interpret(a_1...a_i)(s_0)$$

where the a_1 are inputs and a sequence of displays as $d_1...d_n$ where $(s_i, d_i) \in view$. We will use functional notation to express the same relation $d_i = view(s_i)$. Given any sequence of actions p we want to model how the user might predict by using knowledge of the history of the interaction. Usually not all the history is required; rather, only some components of the history are important. We can define such *historical predictability*. First, we let

$$history(p) = < <p_1,d_1>, <p_2,d_2> .... <p_n,d_n>>$$

where $p = <a_1,...a_n>$ and $p_i = <a_1,...a_i>$ and $d_i = view(interpret(p_i)(s_0))$. The history generates a record of all the information that may be recorded about the system in terms of inputs and outputs at the various stages of the interaction.
A system is *historically predictable* if, for any input sequences p and q satisfying

$$view(interpret(p)(s_0)) = view(interpret(q)(s_0))$$

and

$$history(p) = history(q),$$

we have for any subsequent experimentation using input sequence r,

$$view(interpret(pr)(s_0)) = view(interpret(qr)(s_0)).$$

This notion of predictability assumes perfect memory. A user of the system will find it possible to predict with certainty what is happening only if every detail can be recalled. In practice there is a spectrum between requiring perfect memory and requiring no memory (as in the case of observational predictability). We therefore specify a further notion of predictability that depends on the user extracting the requisite information from the history. We assume a function *extract* and define *partial predictability* in a similar manner.
A system is *partially predictable* (based on a function *extract*) if, for any input sequences p and q satisfying

$$view(interpret(p)(s_0)) = view(interpret(q)(s_0))$$

and

$$extract(history(p)) \;=\; extract(history(q)),$$

then for any subsequent experimentation using input sequence r,

$$view(interpret(pr)(s_0)) \;=\; view(interpret(qr)(s_0)).$$

Here *extract* $<<p_1,d_1>, <p_2,d_2> \ldots\ldots <p_n,d_n>>$ will pick out attributes from the history of the interaction. For example it may define that part of the history that contains the key incidents in the construction of the diagram.

We can learn more about the nature of *extract* from this scenario. As we have said, the selection method described above is clearly not observationally predictable in the sense described above. The operations necessary to select a required object cannot be foreseen unless the user is familiar with the history of how the picture was created. For example, when the user clicks in an area containing overlapping objects, it is not possible on the basis of visual structure alone to predict which object will be selected. Thus, two identical drawings may behave differently. We must now ask in what sense this system is predictable. It is possible to predict the effect of selection if it is known how the drawing was constructed. Predictability is dependent on certain features of the preceding interaction history, namely those features which determine how the drawing was constructed. We have a notion of partial predictability in which *extract* identifies those components of the history that relate to the way in which the system was constructed. The function *extract* can be constructed in steps in this scenario.

First we require a function that operates on input/display pairs to determine if the input in the context of the display represents a significant command in the construction of the graphical objects. We represent this function as

$$construct: (P \times D\,) \rightarrow C\,.$$

$construct(p,d)$ will yield a nontrivial command only if p produces a change to the temporal or structural hierarchy, noticeable in the display d, and that was not noticeable for any prefix of p. If we map *construct* onto the *history*, we will get a trace of the commands that affect the temporal and structural hierarchy.

It is not necessary to remember all of the structural changes that have occurred in an interactive session. For example, objects that have been created then deleted and are no longer recoverable will not affect the future interaction. The user need not remember that they ever existed. Therefore, we can filter out such meaningless commands from the result of mapping *construct* onto the *history*. This completes our construction of the extraction function for this scenario. In summary, we would have

$filter: C^* \rightarrow C^*$

$extract = filter \circ (map\ construct)$

Another possible refinement to this model, which we will not expound upon in this chapter, would be to use attributes and signatures to arrive at a definition for the *extract*ing function. We argue that the complexity of the functions *construct* and *filter* indicate some system-projected measure of the memory load on the user.

3.3. Conclusions

We conjecture that *extract* is a function whose appropriate definition (taking into account more global design implications) may be clarified by a psychological analysis. We can only conjecture at this stage and note that this bridge between psychology and computer science represents an important component of the AMODEUS project (Barnard & Harrison, 1989). This connection is further discussed in the next chapter. Design must capture assumptions about the system's environment (such as, the psychological make-up of the user population, work patterns, organisational architecture and so on). Therefore, embodying psychological findings within a system model is one important means of making explicit the extent to which an interface design can exploit such findings. In Chapter 2.3 we consider examples where user orientated view points cannot be folded into the system model. If we now turn our attention back to the example drawing package and the object selection method, we can analyse what memory demands the system design imposes on the user. In this case, the user needs to comprehend all three hierarchies which determine how the selection is made. The visual hierarchy is by its nature displayed while the structural and temporal hierarchies are not usually visible. Hence, for the selection procedure to be predictable, *extract* will be required to include the structural and temporal hierarchical information (such as an ordered tree of the graphical objects present). The extent to which users are able to reliably remember such information can be the subject of empirical analysis. In this way discrepancies between the mental competence assumed by a system and users' capabilities can be discovered. An analysis of this kind will determine whether it is necessary either to simplify the selection algorithm or to make explicit the structural and temporal information so that the user may derive from the display what has to be known in order to predict how the selection will work. In most graphics packages the selection algorithm is simplified. An appropriate solution to this design problem will depend on the way the system is used, so there may be cases when structural and temporal information should be made explicit as a more valuable alternative to simplification of the algorithm.

4. General Conclusions

We have demonstrated in this chapter how a formal approach to system analysis can focus on the interactive aspects of the system and provide a means for elucidating principles to enhance interaction within the early design stages. We are concentrating our efforts now on incorporating the results of the abstract interaction models within more constructive software engineering and formal notations (Abowd, 1990; Abowd & Harrison, 1990; Duke & Harrison, 1993). We hope that our work will convince practitioners in formal methods of the value to be gained in extending existing techniques into the realm of requirements which on the face of it do not appear to relate to the functionality of the system, of which human-computer interaction forms a significant concern.

5. Acknowledgements

We would like to thank members of the AMODEUS consortium for providing the initial descriptions of the scenarios used in our analysis. Discussions with other, more psychologically oriented modellers have continually led us in the direction of incorporating relevant psychological principles in our formal interaction models. We would also like to thank the members of the York HCI group for their constant support of our work and the stimulating discussions that they have provoked.

6. References

Abowd, G.D. (1990) Agents: communicating interactive processes. In D. Diaper et. al., editors, *Human-Computer Interaction — INTERACT'90*, Elsevier, pp. 143-148.

Abowd, G.D. & Harrison, M.D. (1990) On a constructive approach to applying formal methods in HCI. University of York, Department of Computer Science, Technical Report YCS (151).

Barnard, P.J. & Harrison, M.D. (1989) Integrating Cognitive and System Models in Human Computer Interaction. In A. Sutcliffe & L. Macaulay, editors, *People and Computers V*, Cambridge University Press, pp. 87-104.

Duke, D. & Harrison, M.D. (1993) Abstract Interaction Objects. In *Proceedings Eurographics '93, Barcelona.* Springer.

Hammond, N.V. & Myers, K. (1990) Definition of scenarios for M1 Workshop. ESPRIT BRA project 3066 (AMODEUS) Internal Report RP8/IR6.

Harrison, M.D. & Dix, A.J. (1990) A state model of direct manipulation in interactive systems. In M.D. Harrison & H. Thimbleby, editors, *Formal Methods in Human-Computer Interaction,* Cambridge University Press, pp. 129-152.

Harrison, M.D., Roast, C.R. & Wright, P.C. (1989) Complementary methods for the iterative design of interactive systems. In G. Salvendy & M.J. Smith, editors, *Designing and Using Human-Computer Interfaces and Knowledge-Based Systems*, Elsevier, pp. 651-658.

Payne, S.J. & Green, T.R. (1986) Task-action grammars: a model of mental representation of task languages. *Human-Computer Interaction*, **2**:2, pp. 93-103.

Spivey, J.M. (1988)*The Z Notation: A Reference Manual.* Prentice-Hall International.

Young, R.M. & Whittington, J. (1990) Using a knowledge analysis to predict conceptual errors in text-editor usage. In *CHI'90 Human Factors in Computing Systems*, ACM, pp. 91-97.